INCREASING DESIGN COMMUNICATION USING VIRTUAL REALITY

by

JUSTIN W. MILLS
Bachelor of Architectural Engineering
Oklahoma State University, 1997

Submitted to the Department of Civil and Environmental Engineering in Partial Fulfillment of the Requirements for the degree of

Master of Engineering in Civil and Environmental Engineering at the Massachusetts Institute of Technology

MAY 1999

Signature of Author: [illegible]
JUSTIN W. MILLS
Department of Civil and Environmental Engineering
May 7, 1999

Certified By: [illegible]
JEROME J. CONNOR
Professor of Civil and Environmental Engineering
Thesis Supervisor

Accepted by: [illegible]
ANDREW J. WHITTLE
Professor of Civil and Environmental Engineering
Chairman, Committee for Graduate Students

INCREASING DESIGN COMMUNICATION USING VIRTUAL REALITY

by

JUSTIN W. MILLS

Submitted to the Department of Civil and Environmental Engineering on May 7, 1999 in Partial Fulfillment of the Requirements of

Degree of Master of Engineering in Civil and Environmental Engineering

ABSTRACT

As the world of building design becomes more complex, the need for sophisticated software will increase. A design environment that unifies the consultants and the clients will greatly benefit all involved and will result in better building designs. Using the Virtual Reality Modeling Language, coupled with the Java programming language, the communication of design decisions and design intent can be greatly improved. Simple teaching tools illustrate the effects of design decisions, such as increasing a beam span to cause a beam to increase in size. Once the design is complete, communication with the client and the builders can also be improved using this technology. Instead of having a set of drawings and a set of specifications, a single virtual reality model can be accessed and queried to find out information on specific pieces and the process involved in creation of building systems. Finally, VRML is not an overall solution to the integrated design environment problem, but rather a tool to improve communication of ideas to others involved in the building industry in a convenient, Internet-based form.

Thesis Supervisor: Jerome J. Connor

Title: Professor of Civil and Environmental Engineering

ACKNOWLEDGEMENTS

I would like to thank Professor Connor and Professor Helliwell for their help in the completion of this document. Their time and effort make the High Performance Structures track of the Master of Engineering Program at MIT such a success. I would also like to thank each of my professors who provided me with the technical knowledge needed to complete my studies.

Personally, I would like to thank my family for their support throughout my educational career. Finally, I would like to thank my wife, Jennifer, for her support and understanding, without which I would not have achieved my goals.

TABLE OF CONTENTS

TABLE OF FIGURES

Chapter 1 - INTRODUCTION

Introduction

From our beginnings, man has labored over the design and construction of new buildings. Some of these buildings are built for monumental reasons and others for pure functionality. Although the Egyptian Imhotep is considered the first architect, many others before him labored over how to create structures to protect people from the elements and to give them a sense of home. From simple structures, more complex building were envisioned and some were even built. There exist today structures that were designed centuries ago that cannot be re-constructed, even with today's technological advances. As these designs became more complex, more people were necessary to handle the complex details, and with advancement in the sciences, even more people were needed to design structures. Recently, computers have been introduced, allowing designers to store massive amounts of data and perform millions of calculations on buildings before they are even built. Since the introduction of computers, the profession has been in need of an environment similar to the old style of building design, in which all designers worked from a common location and toward the same goal. Today, the design team is dispersed and often the project is located far from any single designer. The computer lends us the tool to create an environment where all parties involved in the design and building process can meet to visually communicate their ideas to each other.

The scope of this thesis includes the theoretical aspect of utilizing virtual worlds and some real applications. A basic knowledge of the Virtual Reality Modeling Language and the Java language is assumed. The overall concept and structure of each of the examples is discussed and an in-depth explanation of areas specific to Java and VRML is provided.

Chapter 2 – History of Building Design Process

Background

Building design has become a more complex task than in the days of a master architect responsible for all parts of the building design, including the beauty of the building, the strength of the structure and the comfort of the environment. As buildings evolved and codes were established to provide safety to occupants, the single architect was no longer able to accomplish the entire design task alone. Therefore, consultants were brought in to aid in designing specific parts of the building, allowing the architect to focus on the architecture. Eventually, these consultants formed their own organizations with their own guidelines and responsibilities for their segment of the overall building task. Currently there is a trend of specialization within each of these major organizations into further refined consultant groups. The client or general consultant usually requests specialty consultants to solve a specific problem that is beyond their expertise. An example of this is in the area of vibration design in structural engineering. With the increasing complexity in seismic design and the need for motion sensitive buildings such as chip manufacturing plants, specialists are needed to aid the general structural engineer in his task of designing the structure of a building. Another example, even closer to the architect's role, is the acoustic consultant for acoustic-sensitive projects such as theaters. This consultant recognizes the needs of the architect and the goals of the design and provides expertise in an area that the architect may not have.

Unlike other design professionals, those involved in building design don't have the advantage of prototyping their design and testing its effectiveness. Mock-up pieces of the building can be built at full or partial scale to test the component, but full interaction between all the pieces can never be tested until the structure is built. This makes the design of buildings unique from the typical product design process and is a source of problems since mistakes are usually identified and corrected during the prototyping phase. Designing a modern building requires multiple teams of designers to work in unison under the orchestration of a central project manager. The role of project manager is sometimes assumed by the architect and is required to coordinate the efforts of each consultant.

Dispersed Desig n Team

It is rare to find a constructed facility that is designed, engineered and built by the same company. Even when this is the case, the work is typically spread out across multiple offices, dividing the work among the company's resources. More typical is the case where a design team is made up of an architect or central designer supported by several consultants who are geographically dispersed as seen in Figure 2-1. As buildings become more complex and consultants tend to specialize there is a trend toward hiring a specialized consultant for a specific area of the building design process rather than hiring a general-purpose consultant. For example, when a high-tech building is designed, a specialized electrical consultant is required to design the specific hardware wiring, but a traditional electrical engineer may design the standard wiring, including the lighting systems. As several small teams form to apply their skills in various areas of the building design, communication within and among teams becomes strained, yet is essential to success.

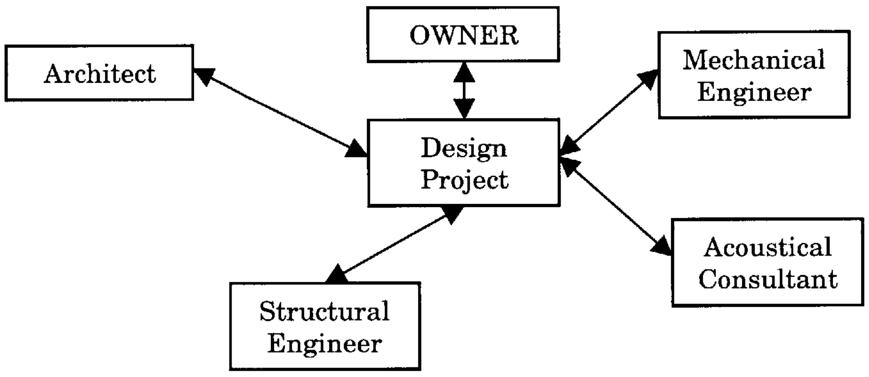


Figure 2-1 Dispersed Team Diagram

Physical meetings have and probably will continue to be the ideal communication scenario in any environment. Pure interaction can occur and nothing is left to guess at, such as a long pause on the other end of the line during a phone meeting. Because creating architecture is so visual, physical meetings are required to thoroughly explain and understand what is being discussed. Artificial meeting environments cannot currently meet many of the needs of those involved in building design. Typically the sheer size of the materials used in the design process, i.e. models, drawings sheets 24" x 36", cannot be shown on a small computer screen. If shown on projection screens, the individual speaking is lost, and can, at best simulate a pointing finger with a mouse movement. All aspects of creating architecture are messy processes, usually requiring access to several large drawings at once. When multiple consultants meet, each with their own set of drawings, a paper war is waged as marks are frantically scratched onto sheets leaving notes for later use. Even with the advent of CAD systems, there is no good way to quickly sketch and annotate CAD drawings in the quick and dirty manner designers are so used to.

Paper-Based CA D

With the introduction of Computer Aided Drafting into the design environment, it would appear as though design professionals have come a long way from the old paper-based documentation. It was the practice to trace and re-trace existing drawings from consultant to consultant. An architect would send his or her plans and details as a blueprint to each of the consultants who would in turn trace these, or even completely re-generate them from scratch. This required hours of manual labor to produce documents that now are seen as works of art because of the workmanship that went into producing them. CAD systems were the solution to this problem by providing an environment similar to word processing systems, where editing and re-editing of existing documents is capable. Using CAD systems the architect draws the CAD plans and sends copies to each consultant. To this point, it is very similar to the old process, only in an electronic format rather than a paper format.

In an ideal CAD process, the consultants merely place their information within the same file, using the architects' plans as a base to their plans. This is where most design teams fail. This is typically done using layers, similar to the layers of tracing paper that litter architect's offices. By breaking up the content of a drawing, universal information such as grid lines and wall locations can be shared by each drawing file. This allows the next set of drawings to start with a base set of information allowing them to remove any unnecessary information that may also be include in the drawing. The real-life process begins with the conversion of the CAD file to a format that is consistent with that particular office's standards, changing line colors and layer naming. Conversion of drawing formats is typically a full time job within large companies. As alliances are made between designers, some companies will generate programs to do this conversion automatically, dissecting each drawing file, and changing it accordingly. This appears strangely close to the old procedure of tracing or even re-drawing the sent document. For this reason, traditional CAD systems will never attain integration between design teams and design information will keep being changed and modified by the current user to match their version of the other consultant's standards.

Integrated Syste ms Solution

Traditional CAD systems have aided the building design profession in the past, but now it is time to take design communication a step further. As the public expects near perfect design, allowing little room for errors, design professionals must take extreme care to simulate and check every area of the building. What better environment to do this than a three-dimensional world where the structure truly can be built (See Figure 2-2).

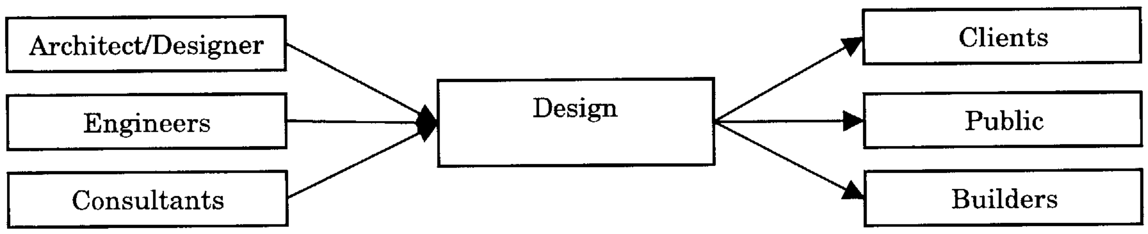


Figure 2-2 Integrated Design Environment

Such systems exist within the mechanical engineering world and provide a networked environment where designers from many locations can edit pieces of the model simultaneously. As soon as a change is made, the entire model must be updated to reflect the new geometry, material property or attribute. Additional components to the base system perform more advanced design calculations and geometry checks. This environment has enabled the mechanical engineering industry to produce physical parts close to their final design the first time they are produced, rather than after several iterations. This is because the CAD environment allows testing of their pieces in many ways to simulate actual tests that will be performed on the physical parts. Basic geometric modeling is the basis for such a system allowing the designer to generate complex three-dimensional models of the building components. This is what is done using traditional CAD systems today, generating solid models from two-dimensional drawings to simulate the actual design. The difference lies in how such models are created.

In the integrated system, the new piece can be constructed based on existing components and specified relative to their geometry. For example, using a traditional CAD system, a designer creates a box and places it on two column boxes to represent a beam. An integrated design environment would accept the input as a geometric shape that spans between two existing objects, therefore, when one of the columns is moved, the beam automatically elongates or shortens to react to the change in geometry. Additional constraints could be added using component design sub-routines that specify the geometry of the beam as a function of the weight of components resting on it and the span.

This system would require a great amount of overhead at the beginning of a project to input all of the geometry and the associated data. The payoff would be in the iterations during design revisions when changes are made to systems and components of the building. During this period, only minor changes will be required, because other changes will occur automatically as a result of the single change. As a detail is changed or updated only that particular detail need be worried about. The reason being that the other parts of the building that rely on that component would have been designed relative the component and thus updated with the new component data.

Another feature of this design approach is the actual model itself, which can be used in presentations to clients, and even passed on to facility-management organizations who may have use for it in building maintenance programs. As the public demands safer and "functional" designs and the systems and components of buildings become more complex and numerous, the need for such systems will be inevitable. Eventually all design may take place in a virtual world where even the site is pre-existing in the model and designers add to the virtual world just as they add to the real world.

VRML Solution

Although still in its infancy, the Virtual Reality Modeling Language (VRML) provides the ideal environment for viewing and simulation within integrated models. VRML is currently supported by Internet browsers via a small plug-in, which can be downloaded and installed free of charge. VRML is a text-based system of describing complex three-dimensional geometry with the capability of motion and user-geometry interaction. Currently creation of such worlds can become cumbersome, requiring a three-dimensional modeling package with VRML output capabilities or great patience to hand code each piece. Using an existing package such as 3D Studio MAX can greatly speed up the process, generating much of the geometry in a rich 3-D design environment. At this point, it is back to text-based programming to key in responses to user actions and add interactivity. With the advent of recent VRML specifications, Sun's Java technology has been added to

VRML97

ISO/IEC 14772-1:1997

Figure 2-3 VRML 97 Logo

VRML, which seems an obvious step since both technologies are viewed on the Web. This addition is in hopes of providing a more integrated design and viewing environment where users can access all the potential of Java with the graphical capability of VRML. Using Java, many more things are possible, such as linking VRML to databases for more efficient data storage, interaction with users through standard window interfaces, and networking between multiple users/designers.

VRML will not be able to replace traditional CAD systems, because VRML is specifically geared toward the Web and graphics applications, and is merely a consistent format for describing three-dimensional geometry. It does not have integrated database support, which is key to efficient data storage and has no efficient way to create complex objects. The addition of Java solves some of these problems, allowing access to databases and other high-level language features. One realm that VRML can be used in is the presentation and documentation of design. Using VRML and Java's JDBC (Java DataBase Connectivity) features, Web users can have access to the actual model file, through views only. This can be used as a teaching tool to show virtual building environments and the implications of design decisions. It can also be used to show the construction process providing a greater level of detail that construction documents will never attain. On the client side, this VRML environment will show the building from a user perspective, providing information about specific features.

Although linking a VRML file to an integrated design environment is beyond the scope of this paper, attempts will be made to show the potential of VRML in the area of design communication. The VRML specification is a designed to be platform independent although the actual support is in the individual browsers used to convert the VRML source file into the virtual world. In summary, VRML does provide an environment that can be used to show potential features of a fully integrated design environment and the improved communication it will accomplish.

Chapter 3 – LEARNING SYSTEMS

Introduction

Communication between design professionals is crucial to any project. In a typical project, the architect coordinates communication between the engineers, architects and other consultants. One issue that plagues design teams is the implication of each of their decisions on other team members. Even the slightest change of dimension on a particular item can have major consequences, especially given the increased accuracy of current design and construction techniques. Each design professional involved needs to understand the other systems in the building and how they must all interact for the building to perform as a whole. A single model design application would solve this problem, allowing one team member's model changes to register on other members' models. VRML tends to be the least applicable in the design phases since those involved at this level require very powerful systems with the highest degree up adaptability. However, VRML is a powerful format that is quite portable and can be used in a scaled-down version of a fully integrated design environment. Since the VRML format is easily available on the Internet with a wide range of plug-ins to many browsers, virtual worlds can be an effective tool to communicate design information between student and teacher. Typically in the teaching environment, the examples discussed are only a portion of the scale of problems seen in the workplace. Based on linking VRML with a programming language such as Java, some components of an integrated design environment can be accomplished.

Space Relations hip Example

Although not a direct benefit to those involved in the design of actual structures, a teaching tool to show the effects of design decisions can be implemented in VRML utilized a Java scripting node. To illustrate the very basic features of Java and VRML, I will begin with an example of two rooms, directly adjacent to one another (Figure 3-1). Then, I will provide the capability to move the dividing wall back and forth, demonstrating that by deciding to make one room bigger, another must get smaller.

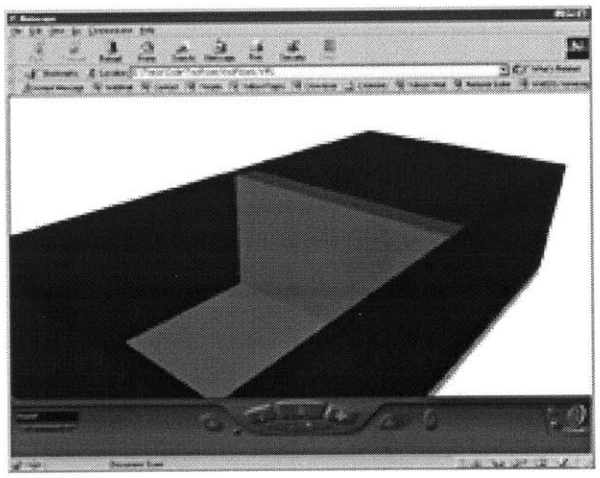


Figure 3—1 Virtual Reality Model of Two Rooms Example

This example may seem trivial, but is directed at explaining the basics of using Java and VRML in an example showing design decision implications. The goal is two rooms with a single interior partition between them. Using VRML, apply sensors to each of the two sides of the interior wall and then associate the sensors with a Java program. Using Java, it is possible to control the movement of the two faces of the interior wall to move in unison, away from the side of the wall that was clicked.

The first step is to create the basic geometry using 3-D Studio Max, composed of a floor, four walls and two faces of an interior wall. The interior wall is composed of two pieces to separate them from each other to allow for two sensors within the VRML world. Once in VRML format, the .wrl files are incorporated into a Visual Studio project, which contains the Java programs. Once in this environment, *TouchSensor* nodes are added to the *Transform* nodes:

```
DEF InteriorWall2 Transform {
  translation 5 0 0
  children [
      Transform {
        translation 0 50 0
        children [
          Shape {
            appearance USE int_app
            geometry Box { size 10 100 180 }
          },
          DEF Wall2Sensor TouchSensor {}
    ] }
  ]
}
```

The *TouchSensor* node converts the entire object into a sensor, making the entire wall a sensor, which allows the user to select the object by clicking anywhere on the wall, even on the top of it. The *TouchSensor* sends two major events: one when the user positions the cursor over the node and another when the user clicks. Route the *isActive* event, which occurs when the mouse clicks on the object, into the event handler method of the Java class. The next step in the process is the addition of the *Script* node, which defines the location of the Java .class file and any inputs, outputs and extra information to send from the VRML file to the Java program.

```
DEF OneRoomControl Script {
      directOutput TRUE

      field SFNode w1 USE InteriorWall1
      field SFNode w2 USE InteriorWall2

      eventIn SFBool clicked1
      eventIn SFBool clicked2

      url "moveWall.class"
}
```

The first value set is the *directOutput* field of the *Script* node. By setting this to true gives the Java application the ability to directly output values to the VRML file. This will be useful when we begin moving several objects, and we want them to be moved directly from the Java application rather than passing an event back to the VRML source file and then routing that event to the corresponding *translation* and *scale* nodes. The *field* labels define information that we want access to from the Java application. In our case we want to pass a reference to the two interior walls so we specify that the variable use the existing nodes *InteriorWall1* and *InteriorWall2*. We must also specify the data type of variables in order for Java to be able to use the information contained within the variable. In our case, we will be passing a VRML node(*SFNode*), which we will use to access the *translation* field to set the position.

The next field modified is the *eventIn* and the *eventOut* fields that direct the input and the output to and from the Java application. The eventIn fields are the actions that will be used to evoke the Java application, while the *eventOut*'s are the output from the application back to the VRML file. Since we have chosen to enable direct output to the VRML source file, we will not need any *eventOut*'s to change the values of the various nodes. Next in line is the URL or Uniform Resource Locator of the Java .class file. The final step within the VRML source file will be the routing of events from the *TouchSensor*'s to the *eventIn*'s of the Script nodes.

```
ROUTE Wall1Sensor.isActive     TO OneRoomControl.clicked1
ROUTE Wall2Sensor.isActive     TO OneRoomControl.clicked2
```

Notice that the two *eventIns* that were defined in the *Script* node are on the receiving end of the *TouchSensor*'s isActive *eventOut*'s. These events are of a Boolean type, true or false, true when they are touched and false when they are released. This will be important when we trap the events coming into the Java application because we do not want to move the wall twice, once when the mouse is clicked and again when the mouse is released.

Moving on, the Java class file that contains the methods and properties necessary for all the movement of the interior wall. In order to access any of the VRML nodes and their associated properties, several packages will be imported. The vrml.node packages contains the Script class, which the Java class must extend. This package must be included in every application that is going to be called from the VRML *Script* node. The other packages imported are the vrml and the vrml.field that allows access to the VRML field types such as *SFBool* and *SFVec3f* and access back to the VRML world and the viewer.

```
import vrml.*;
import vrml.field.*;
import vrml.node.*;

public class moveWall extends Script{
      private SFNode w1, w2;

      public void initialize() {
                  //Connect java nodes with vrml nodes
```

```
                w1 = (SFNode) getField("w1");
                w2 = (SFNode) getField("w2");
        }

        public void processEvent(Event e) {
                ConstSFBool v = (ConstSFBool)e.getValue();
                //Only perform the movement on mouse click, not
release
                if (v.getValue()) {
                        SFVec3f t1, t2;
                        t1 =
(SFVec3f)((Node)w1.getValue()).getExposedField("translation
");
                        t2 =
(SFVec3f)((Node)w2.getValue()).getExposedField("translation
");
                        if (e.getName().equals("clicked1")==true)
{
                                t1.setValue(t1.getX()+10.0f,
t1.getY(), t1.getZ());
                                t2.setValue(t2.getX()+10.0f,
t2.getY(), t2.getZ());
                        }else if
(e.getName().equals("clicked2")==true) {
                                t1.setValue(t1.getX()-10.0f,
t1.getY(), t1.getZ());
                                t2.setValue(t2.getX()-10.0f,
t2.getY(), t2.getZ());
                        }
                }
        }
}
```

Once inside the class, we have access to the VRML data types, allowing us to define variables such as *w1* and *w2* that are of type *SFNode*. The *initialize*() method is called when the VRML browser load the .wrl file just as a constructor would be invoked during a new object creation. In this method, we get the field values defined in the script node of the VRML file that reference the two nodes we need: interior walls one and two. We specify the data type of the fields to convert them into the Java data types that are equivalent to the VRML data types. These variables will be used to reference the nodes directly to gain access to any of their properties, specifically the *translation* field that contains the node's position.

The *processEvent* method is evoked each time any event is routed into the script node, passing the event that called it as a parameter. Since we know all input events are from *TouchSensors*, we immediately grab the value of the event, so we only execute during true values corresponding to mouse clicks ignoring all false mouse release events. Once we have decided to execute the movement, we use the reference to the VRML node to gain access to the translation field that is a three-dimensional vector data type.

Now that we have a reference to the actual location in the VRML file where the position is stored, it is merely a matter of determining which wall has been clicked and changing the value of the translation accordingly. We will use the *getX*, *getY* and *getZ* methods of the variable that points to the translation field to set the new value relative to the current position. Using this, the Java class file can be used again with any other two walls, defined in a similar matter placed anywhere in space. The only restraint is that motion will only take place in the X direction.

Once implemented, the once static VRML world now becomes dynamic allowing geometric changes simply by clicking on the interior wall objects. Three views have

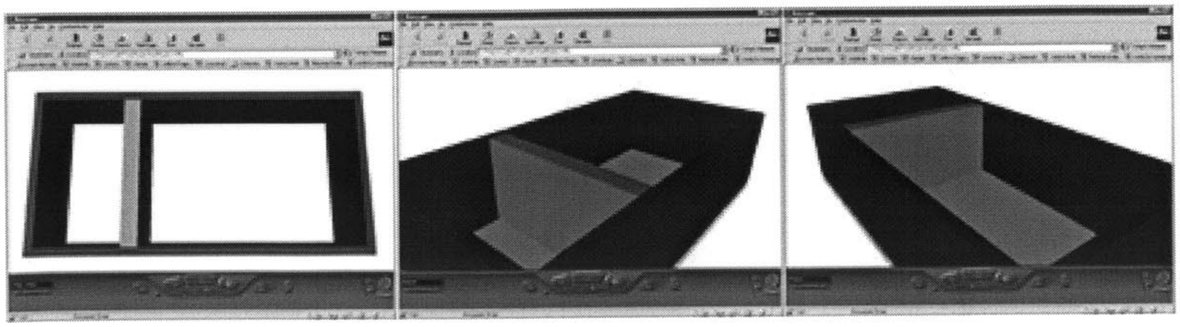


Figure 3—2 Plan, Room1 and Room2 Views

been set up, giving positioning to move the wall in either direction and to view the overall effect the wall position has on the two spaces (See Figure 3-2). Figure 3-3

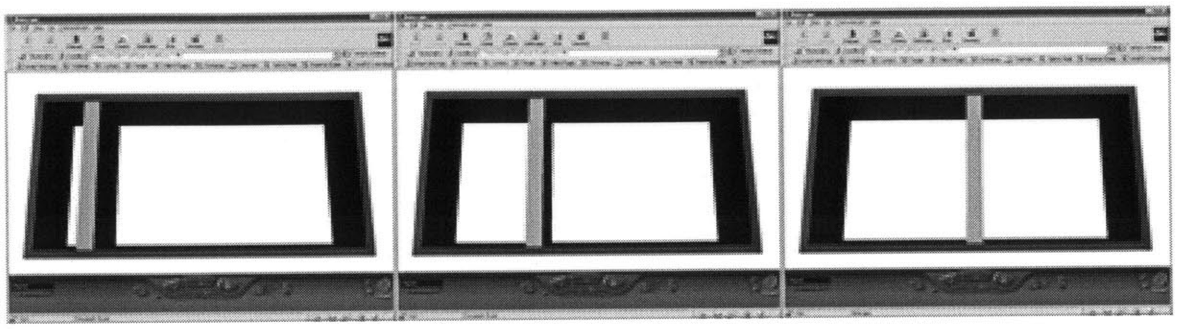


Figure 3—3 Plan Views of Two Rooms World

shows various configurations of the wall within the space, ranging between two square rooms to a hallway type room and a rectangular room. The source files for the VRML world and the Java application can be found in Appendix A as with all others in this paper. Now that we have some of the basics of the interaction between Java and VRML, let's use the computation associated with structured programming languages to provide intelligence in moving and scaling VRML world items.

Simple Beam Co lumn Example

As discussed, an interactive design environment in which the designer's decisions affect all other components in the model can be a powerful tool during the design process. Designing in a parametric environment where beams are specified as spanning between two points and carrying specific loads can greatly increase efficiency. If the designers choose to make changes to the model, such as moving a beam or removing it altogether, the rest of the model can be automatically updated. This is similar to performance-based specifications that are gaining popularity in the building design community.

Using performance-based specifications, the designer requires that a component of the design meet certain requirements such as the stress shall be less than A or the deflection under service loading shall be less than B. Many view this as a way to shortcut designing all components in the building, but it is more of a necessity in modern complex buildings. As buildings become more technologically advanced and the materials used to construct these buildings become more complicated, designers are required to be experts in more and more areas. This results in designers that have a general knowledge base, but have no level of expertise in any one area. These designers are under-qualified to design complicated uses of building materials or handle complex situations. For example, if an engineer who has only designed steel buildings for twenty years, was asked to design a concrete high rise, the engineer may proceed to analyze the structure just as though it were a steel structure, only substituting in the material differences. This may result in a dangerous design because a concrete building behaves quite differently than a steel building. For this reason, the engineer may find it useful to employ a concrete design specialist, minimizing his liability and producing the best possible final result.

In an interactive design environment, the computer application can be used to provide the expertise of a specialist consultant. For instance, the designer specifies requirements, such as the size of a space, while the consultant specifies the relationship between the volume of a space and a particular component such as duct sizes. It may appear that a specialist will no longer be needed using this environment, but will be necessary to specify how the design is computed.

Therefore, the design of a building will become more like developing a software package. A building will have one set of inputs, the geometry and the components of the building, and one set of outputs, the documentation allowing the building to be built.

Many of the everyday programs can be re-used from design to design, such as the placement of a new steel beam in the structural model. Each routine can have specific requirements, such as the beam must span two support points and other geometrical items can be placed on top of it. The engineer can program the beam design or store the beam as an element and use it later in a three-dimensional analysis program to size the member. Programs exist to manipulate geometric information of a design based on the adjustment or addition of new elements, however these programs typically lack the engineering aspect behind the modifications, and thus lack some of the features needed in a building design environment. With the use of Java as an application layer above VRML, such an environment can be shown, but the creation of the objects themselves and the programming behind each element must be hard-coded prior to running. Using Java and VRML, this environment can be used as a teaching tool to students to show real-time design changes and their effects on other elements within the building design.

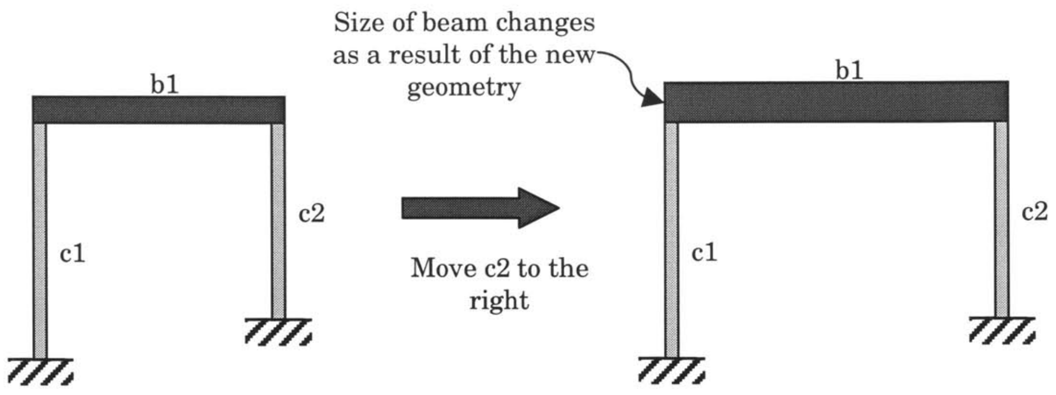


Figure 3-4 Effects of Design Decisions

The previous example was quite simple, but useful to show the basics of connecting a VRML world to a Java program. As a more complex example, consider a simple span beam with a uniformly distributed load along the entire length. All objects within this world will be modeled as simple box-shaped objects for simplicity of shape and ease of beam calculations. As the span is increased or any of the design parameters are changed, the dimensions of the beam are computed and the model is updated as seen in Figure 3-4. The calculations involved in sizing the beam are based on three controlling conditions: shear failure, bending failure and deflection limitations. For a rectangular cross section member, the equations are:

Shear $$\tau = \frac{3V}{2A}; \qquad V = \frac{wL}{2}; \qquad A = bh; \qquad \Rightarrow \tau = \frac{3wL}{4bh}$$

Moment $$\sigma = \frac{M}{S}; \qquad M = \frac{wL^2}{8}; \qquad S = \frac{bh^2}{6}; \qquad \Rightarrow \sigma = \frac{3wL^2}{4bh^2}$$

Deflection $$\delta = \frac{5wL^4}{384EI}; \qquad I = \frac{bh^3}{12}; \qquad \delta = \frac{60wL^4}{384Ebh^3}$$

The user inputs all values except those involved in the geometry of the beam, b and h. For simplification, the height to width ratio is input in the above equations, so they can be solved for b in terms of input values. The height can then be solved using the ratio specified by the user. The deflection input is a limit on the deflection as a function of the span, limiting the deflection to the span divided by some limiting number. Rearranging the above equations, three equations solving for b are used to calculate three widths, of which the maximum can be selected.

```
//check shear
b1 = (float)Math.pow((3 * w * l) / (h2b * 4 * tau), (1.0 /
2.0));
//check bending
b2 = (float)Math.pow(((3 * w * Math.pow(l, 2)) / (4 *
Math.pow(h2b, 2) * sigma)), (1.0 / 3.0));
//check deflection
b3 = (float)Math.pow(((60 * w * Math.pow(l, 3) * delta) /
(384 * E * Math.pow(h2b, 3))), (1.0 / 4.0));
```

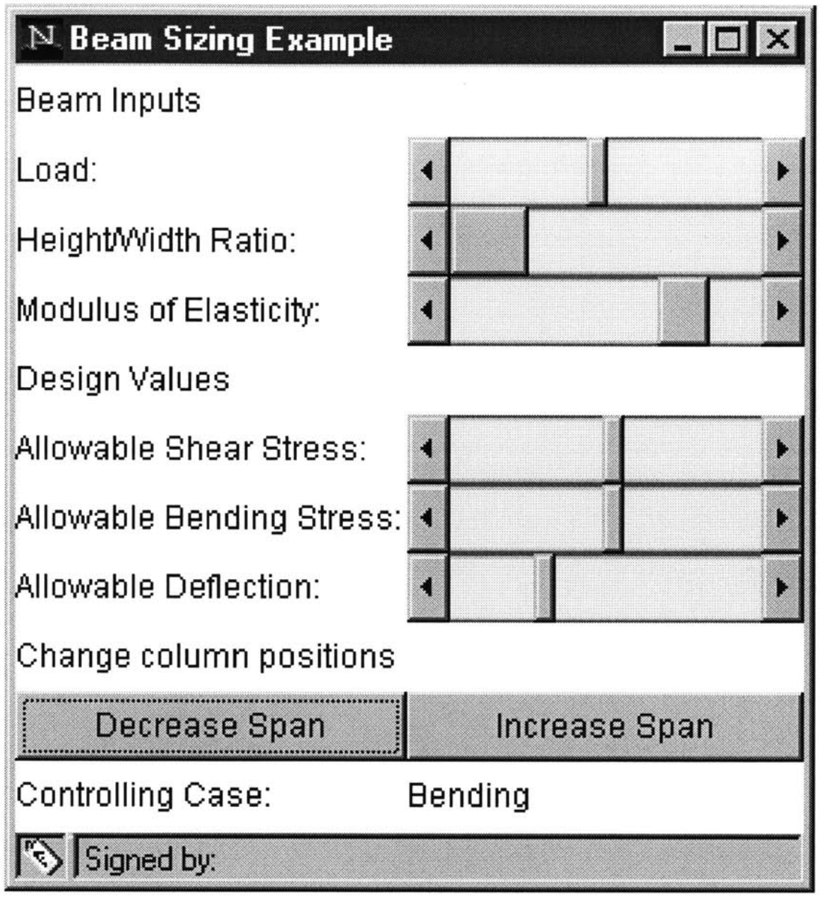
Utilizing the power of the Java Abstract Windowing Toolkit package, an interface has been employed in this application to provide additional user interaction. This technique removes the need for event routing from the VRML source file to the Java application. Only one event is routed to the Java file to provide a way for the user to show the Java window if it is closed or it goes behind the browser.

```
DEF SizeSpan Script {
      field SFNode c1 USE Column1
      field SFNode c2 USE Column2
      field SFNode b1 USE Beam1

      eventIn SFBool size

      directOutput TRUE

      url "sizeBeam.class"
}

ROUTE sizer.isActive          TO SizeSpan.size
```

Most of the above is similar to the previous example. The fields are used as references to the columns and the beam. Since the Java interface will be used to provide user interaction, no *TouchSensor* nodes are needed to cause motion. Once in the Java file, some concepts have been applied, including the new class *trNode* and the class *Binterface*. The *trNode* class is used to provide a more convenient access to the nodes within the VRML environment. Using these classes, many functions can be created to interact with the VRML data and manipulate objects in a more familiar way.

```
import vrml.node.*;
import vrml.field.*;

public class trNode
{
      private SFNode n;
      private SFVec3f t, s;

      public trNode(SFNode nn) {
            n = nn;
            t = (SFVec3f)
((Node)n.getValue()).getExposedField("translation");
            s = (SFVec3f)
((Node)n.getValue()).getExposedField("scale");
      }
      public SFVec3f getTranslation(){}
      public SFVec3f getScale(){}
      public void setlocation(float x, float y, float z) {}
      public void setscale(float xs, float ys, float zs) {}
      public SFVec3f getdist(trNode n2) {}
      public void moveX(float f) {}
      public void moveY(float f) {}
      public void moveZ(float f) {}
      public void scaleX(float f) {}
      public void scaleY(float f) {}
      public void scaleZ(float f) {}
}
```

In the constructor, references to the *scale* and the *translation* fields are established to allow easy access to the VRML geometry. Additionally, several methods are provided to move and scale the object as a function of the existing size and position. The main program uses these methods when columns are moved since their position should only change relative to their existing position. Additionally, the *getdist()* function provides a method to find a vector distance between two *trNodes,* which can be useful in determining the span of the beam. The return value is a vector with the three components (x, y, z) of the vector from *trNode1* to *trNode2*.



Controlling Case: Bending

Figure 3—5 Beam Sizing World Interface

The window class used to add interactivity to the VRML world is a typical windowing class and extends the *Frame* class. In this class, *Binterface*, inputs are gathered from the user and utilized to calculate the beam size. As seen in Figure 3-5, the inputs are broken down into Beam inputs and Design inputs. Following the inputs are the two buttons allowing the user to increase and decrease the span and a text display which tells the user which case is the controlling case. As the beam span is changed, the text is updated based on the requirements for the beam geometry. The window can be closed and can be recalled by clicking on the beam in the VRML world.

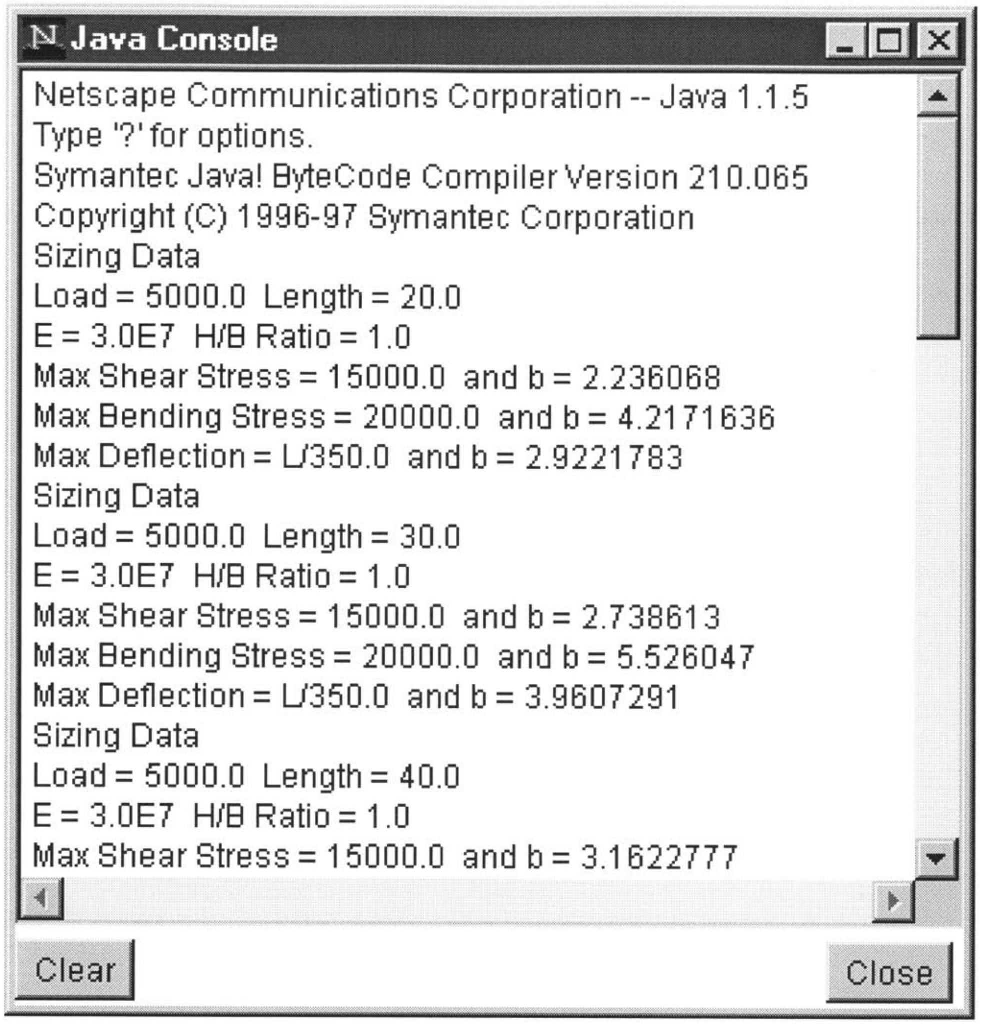
The listener methods required to collect user inputs with the scrollbars and the two interface buttons are located in the main program, *beamSize.java*. The *Scrollbar* listener only recalls the *resize* function since there is no need to change any other geometry in the VRML world. The *Button* listener, on the other hand, needs to determine which button was clicked and move the columns accordingly. Additionally, the *resize* function needs to be called to change the beam geometry. At the heart of the *beamSize* class is the *resize* function that performs all the engineering on the beam and returns the output to the VRML world. The contents of this function are listed below:

```
public void resize() {
	float w, l, E, h2b;
	float tau, sigma, delta;
	float b, b1, b2, b3;
	w = bint.getLoad().getValue();
	l = (col1.getdist(col2)).getZ();
	E = bint.getModulus().getValue();
	h2b = bint.getRatio().getValue();

	tau = bint.getShearStress().getValue();
	sigma = bint.getBendingStress().getValue();
	delta = bint.getDeflection().getValue();
	//check shear
	b1 = (float)Math.pow((3 * w * l) / (h2b * 4 * tau),
(1.0 / 2.0));
	//check bending
	b2 = (float)Math.pow(((3 * w * Math.pow(l, 2)) / (4 *
Math.pow(h2b, 2) * sigma)), (1.0 / 3.0));
	//check deflection
	b3 = (float)Math.pow(((60 * w * Math.pow(l, 3) *
delta) / (384 * E * Math.pow(h2b, 3))), (1.0 / 4.0));
	if (b1 > Math.max(b2, b3)) {
		bint.writeCase("Shear");
	}else if (b2 > Math.max(b1, b3)) {
		bint.writeCase("Bending");
	}else if (b3 > Math.max(b1, b2)) {
		bint.writeCase("Deflection");
	}
	b = Math.max(b1, Math.max(b2, b3));

	System.out.println("Sizing Data");
	System.out.println("Load = " + w + "  Length = " +
l);
	System.out.println("E = " + E + "  H/B Ratio = " +
h2b);
	System.out.println("Max Shear Stress = " + tau + "
and b = " + b1);
	System.out.println("Max Bending Stress = " + sigma +
"  and b = " + b2);
	System.out.println("Max Deflection = L/" + delta + "
and b = " + b3);

	bm1.setlocation(0, 100 + b * h2b / 2, 0);
	bm1.scaleX(b/2);			//Width of beam
```

```
        bm1.scaleY(b * h2b / 2);          //Height of beam
        bm1.scaleZ(1/2);                  //Respan beam
}
```

The first thing this function does is to identify the input values from the window according to the *Scrollbar* values. Next, the three required widths are computed based on the earlier equations. These are compared to determine the worst case, which is written to the window for user interpretation. The maximum width is then set and used to move and re-scale the beam in the VRML world. Additional output is provided in the format of a series of system prints that will print to the Java Console within Netscape Navigator 4.5. This output is used to check the computation behind the world and provide a relative difference between the three cases. The only other function in this class is the *processEvent()* function to capture the VRML *eventIn,* which is being used to show the window interface only.

Use of the interface is quite straightforward. As the world loads, the interface appears (Figure 3-5), allowing the user to begin changing the components. The two buttons provided increase and decrease the span, as discussed. Inputs are in the form of scrollbars and, for the purpose of this exercise, are unitless. The output of each successive change in configuration can be seen in the Java Console window of Netscape Navigator (Figure 3-6). Increasing and decreasing each of the values modifies the data in the model and the result can be seen at the interface bottom where the controlling case is displayed. This environment allows each viewer to position himself or herself in the exact position they so desire. Allowing students to control the view results (Figure 3-7) in a customized learning environment increasing the effectiveness of teaching.



```
Netscape Communications Corporation -- Java 1.1.5
Type '?' for options.
Symantec Java! ByteCode Compiler Version 210.065
Copyright (C) 1996-97 Symantec Corporation
Sizing Data
Load = 5000.0  Length = 20.0
E = 3.0E7  H/B Ratio = 1.0
Max Shear Stress = 15000.0  and b = 2.236068
Max Bending Stress = 20000.0  and b = 4.2171636
Max Deflection = L/350.0  and b = 2.9221783
Sizing Data
Load = 5000.0  Length = 30.0
E = 3.0E7  H/B Ratio = 1.0
Max Shear Stress = 15000.0  and b = 2.738613
Max Bending Stress = 20000.0  and b = 5.526047
Max Deflection = L/350.0  and b = 3.9607291
Sizing Data
Load = 5000.0  Length = 40.0
E = 3.0E7  H/B Ratio = 1.0
Max Shear Stress = 15000.0  and b = 3.1622777
```

Figure 3—6 Java Console

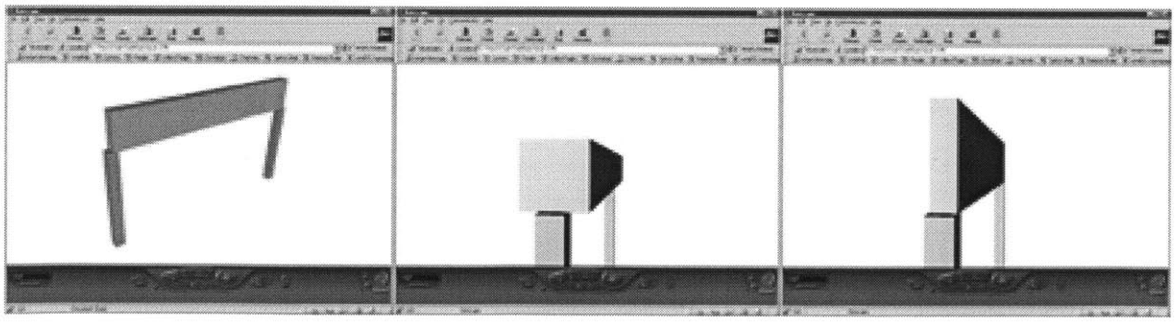


Figure 3—7 Various Views of Beam Size World Showing Effect of Parameters on Section Profile

Chapter 4 – Documentation Applications

Introduction

The uses of virtual reality can be extended into the workplace with simple modifications to existing practices. The communication between designers and builders is key in every project. Any way communication can be improved will most likely be appreciated by everyone involved in the design process. Currently, architects and engineers are limited to communication via two-dimensional drawings and ideas they can voice to others. Providing an additional communication medium gives designers supplemental tools to convey their ideas. In order for this medium to become standard, the generation needs to be streamlined and designers need to become more familiar with it.

There are already capabilities to generate VRML worlds from AutoCAD 14 and 3D Studio as well as many other drafting and solid-modeling programs. At this point, the VRML world can be used to supplement the design information, providing a model for others to investigate details and key components. By utilizing applications within the VRML world, additional information can be attached to the world, providing the viewer with an extremely information-rich environment, including all the information typically found on construction document drawings. Utilizing this format motion can be added to static details, thus illustrating the construction sequence of a building.

Design Details

Using a traditional paper-based delivery system, the builder is usually given two-dimensional drawings of buildings to build three-dimensional structures. Occasionally, isometric drawings will be added to clarify a complex detail or show all components of a system, such as an interior wall. Once in the hands of the builders, these drawing are dissected and annotated further to indicate construction-specific notes, and in some cases, construction drawings are generated solely for construction purposes.

The representation of three dimensions on two-dimensional drawings often causes confusion during construction, resulting in various requests for information (RFI's). A typical problem is a miss-representation within two-dimensional drawings, especially when complex geometry is involved. To solve this problem, many designers create three-dimensional computer models to solve the geometry issue, then create two-dimensional drawings from the model. Although the model often provides checks to prevent common mistakes made during drawing construction, it is wasted because it is not utilized throughout the entire design and construction process. The builder could effectively utilize a three-dimensional model much more than two-dimensional drawings. With a model, the builder can generate any drawing from any view, with access to the exact information needed. Another feature is the ability to access information directly, without the need to produce physical documents. As the modern way of business become more and more paperless, the design environment can adapt its current presentation, transport and storage of documentation to conform. Currently, some limits apply to this methodology, for example, a steelworker on the top of a building will not carry a small portable CAD display window to find out specific details to save one piece of paper.

As an example of VRML's application to construction detailing, consider a typical component of any steel-framed building: the frame. The model for the frame is created in AutoCAD 14 to maintain

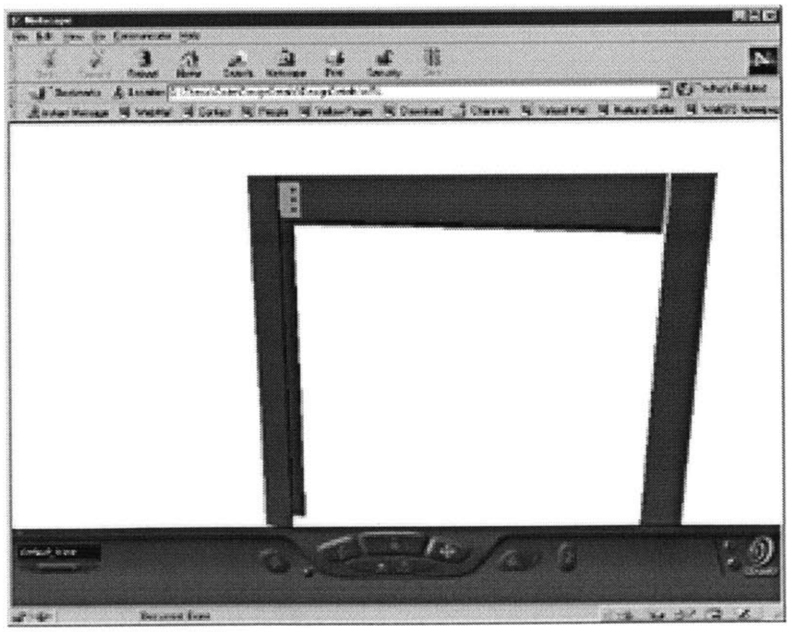


Figure 4—1 Design Details Model

accuracy of the steel sections and components. Two columns are constructed and a simple beam is framed in between and connected by simple shear plates and three bolts at each end as seen in Figure 4-1. Although this detail is not complex, it shows one important feature of VRML: the dynamic display of information. Information is a major concern for designers when constructing design drawings; how much information to display is often determined by the scale of the drawing and the size of the font in order to maintain the readability of the drawings. Using the VRML environment, it is possible to display an information window with any amount of information about any component of the world.

The example will display some predetermined information about different components of the detail, although Java is capable of connecting to text files or databases to access this information. If this VRML detail was generated from a database of design information, the Java *Script* node would read the extended data within the same database to display all pertinent information to the builder or anyone who requests information. This level of accessibility provides the builder with the same level of detailed information that each design professional has, without having to access three physical sets of drawings: architectural, structural and mechanical.

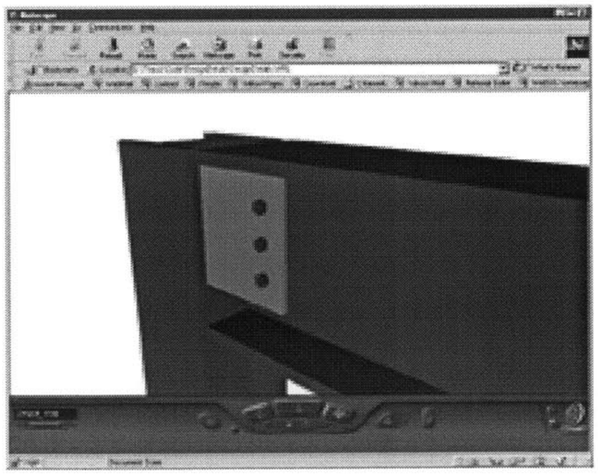
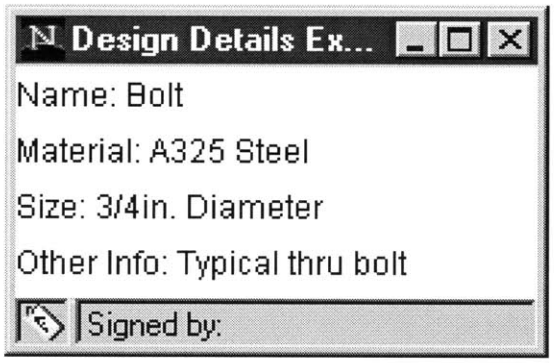


Figure 4—2 Detail of Joint

Notice that when the cursor moves over various pieces of the VRML world as shown in Figure 4-2, including bolts and plates, information is displayed in the Design Details window (Figure 4-3). Although the information is very simple and limited to structural data, all of the information for an entire building could never be provided in one location. Even if this were just the structural model, the amount of information on a single drawing would make it unreadable, while that same amount of information in the VRML world, in merely a click away. To accomplish this, a Java class is created to extend the Frame class providing the window of information. As the user interacts with the elements in the model, the window displays information about the item.



Name: Bolt

Material: A325 Steel

Size: 3/4in. Diameter

Other Info: Typical thru bolt

Signed by:

Figure 4—3 Information Window

The example used is a simple steel frame composed of two columns connected by a beam. Each end of the beam is connected to the column by a single shear plate and three bolts. This type of detail would be provided by the engineer and refined by the steel erector with the final dimensions. Using this information, a contractor would erect the steel structure based on a three-dimensional model where all the necessary information could be easily accessed. The typical information found on erection drawings accompanying an order of steel would be greatly accentuated when applied to a three-dimensional model with geometrical data as well as piece names and overall dimensions.

This example utilizes the same process for Java and VRML discussed in Chapter 3. The VRML file utilizes *TouchSensors* on every element in the VRML world, all of which are routed to the Java file; the processing is quite straightforward. Each event is tested as it enters the Java file. Depending on the source of the event, bolt, beam, or column, predetermined information is routed to the Java window about the particular element. If the full power of Java were used, the Java Database Connectivity (JDBC) package could access a database over a network to query information about the model, therefore allowing a model constructed of old data to be updated and received by the viewer on a real-time basis. Virtual reality provides the next level of visualization in the construction documentation process, giving the viewer control of what is seen and how it is seen.

Process Animati on

Static models of construction details provide several advantages over the traditional two-dimensional detail included in construction documents. One dimension is still excluded: time. Including time incorporates motion into the virtual reality model, responding to user interaction. Typically, when modeling buildings, motion is rarely discussed for good reason: buildings aren't supposed to move! Any structural engineer will tell you that although a building really does move, there is no reason to show a motion in the model. The period of time before construction of a building is finished is a very motion-oriented process. Each building element must be transported to the site and then distributed according to its final location and timeframe. This process is not new to those involved in the construction industry, so why illustrate it to those who see it every day? The reason lies in the fact that as technology advances, so do the components of a building.

With their higher performance level, building components demand additional requirements for their placement with respect to other components. Construction process information is included in the specifications given to builders, whose responsibility it is to correlate that information with the information in the drawings. Although some information is best left in the specifications, such as material specifications for concrete slump and steel strength, other information, such as how to assemble complex features, can be incorporated into a virtual environment. For instance, in a building using viscous dampers to reduce the motion effect of dynamic loads, special care should be taken installing the dampers to insure correct performance. The manufacturer of the damper provides guidelines for their use and construction information, but if detailed animations were produced to show the damper connections, designers could incorporate the animations into the building model. Combination models would give builders the exact process as it applies to the particular building and not just a generalized procedure that can be easily misinterpreted. It will also force designers to address the interface of a specialty item handled by an outside consultant with their standard design components. Catching potential problems at this interface will result in smoother construction and allow designers to solve the problems without requiring any physical change to a partially constructed building.

A review of the history of construction documentation shows an increase in details and information. The necessity of more detailed information require designers to think more about the construction process preventing builders from having to manipulate interfaces between the many building systems, therefore focussing on actually getting the building constructed. This may seem to relieve the construction industry of responsibility, but the fact remains that catching a mistake before construction has begun is ideal compared to finding mistakes with special components at the first floor after ten stories of a building is erected. This area of the design process is quite open to those involved in construction because they generally have a better idea of how field construction is performed and can aid the designers in their decisions.

In a similar manner to the example discussed earlier, information can be attached to the pieces in the animation to describe the process-taking place, such as tightening bolt to specified torque, etc. Similar to linking information to the database, information within a VRML world can be linked to project management software to reference dates that large components are to be completed. If additional information can be provided in the scheduling program, the generation of process animations may be automatically generated for larger pieces of the building, such as the substructure, superstructure, enclosure, interiors, etc. This form of output can also be a very effective form of communicating design ideas to a client when discussing the future construction of a project.

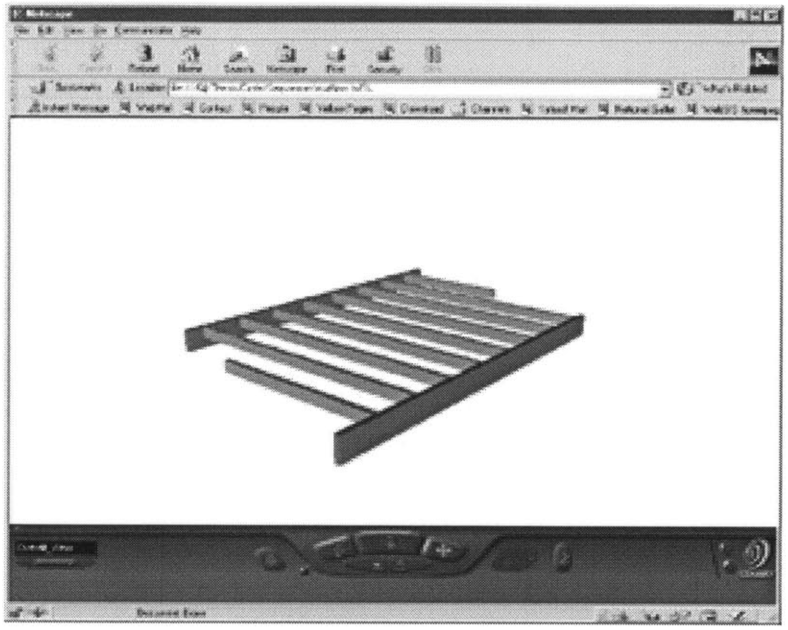


Figure 4—4 Sequence World

To illustrate this use of virtual reality, consider a simple wood floor-framing scheme consisting of two beams with floor joists at sixteen inches on center as seen in Figure 4-4. Plywood is place on top of this and secured using nails. While this is an easy process to describe and most builders know how to do it, the familiarity with the process will allow the example to focus on the method of using virtual reality. The geometrical model is created in AutoCAD using solids. First the beams are created, followed by the joists, which run from face to face of the beams. At the end of each joist is a joist hanger which is attached to the beam prior to placing the joists. On top of the joist, just before laying the plywood, is a bead of caulking used to prevent the floor from squeaking when it is loaded. On top of the flooring system is a series of four feet by eight feet plywood panels that are attached with nails. On top of this system, the architect places the sub-flooring system followed by the finishing material.

The example model focuses on the structural model, although other models can be incorporated into the sequence at any time. From AutoCAD, the model is exported to 3D Studio where the animation of the components is completed. Within the animation time scale, each piece is locked at a specified time depending on the sequence of construction. Prior to the final, locked position, the elements are moved to another location within the model, to represent their storage away from their final position within the model. The sequence of this animation is verified using the previewing capabilities in 3D Studio, ensuring the process is proceeding as planned. Following process approval, the model is exported to VRML format where the addition of the *scripting* node and the Java application creation begins. Within the VRML source file, several fields are added to allow access to the *TimeSensor* nodes from the Java application. This is used to stop the animation by setting the *enabled* field of the *TimeSensor* nodes to false.

In the Java application, as seen in Appendix A, a Frame is created to display the pause button, allowing the user to stop the animation (See figure 4-5). This functionality can be accomplished using sensors within the VRML environment, however, using Java, the sensor can be removed from the world and additional information can be displayed about the component being constructed. A *Boolean* variable is declared to determine the state of the world, either paused or running. An array of *Nodes* is initialized to point to the various fields specified in the *Script* node of the VRML file, which is done to allow looping through each of the nodes, turning them off one at a time, as can be seen in the button's listener method. As the button is clicked, the application tests the state of the world, paused or running, and either resumes or stops animation. Using the

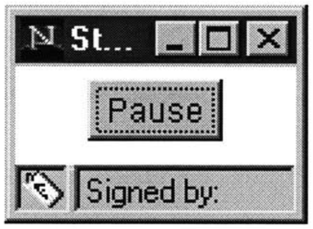


Figure 4—5 Pause Button

array to hold all of the *Node* references enables a loop through them all, setting their *enabled* fields to false, regardless of what they actually point to.

While browsing the world, the pause button in the floating window can be clicked to examine the semi-complete state. The lag between the time the pause button is clicked and the time the animation stops is partly due to the use of Java to listen and respond to the input to directly manipulate the VRML world, which the browser must then update. With the ability to stop the sequence at any time, the viewer has control over the information they receive. This may be particularly useful if the viewer is having problems at a specific phase of construction or they wish to study each sequence as in Figure 4-6. If careful studies could be made of a partially-completed sequence, then builders would be at an advantage as they approached the next phase of the construction. Additionally, smarter controls can easily be built into the control panel of such sequences to further refine the animations and better streamline the information flow that the user needs at any particular time.

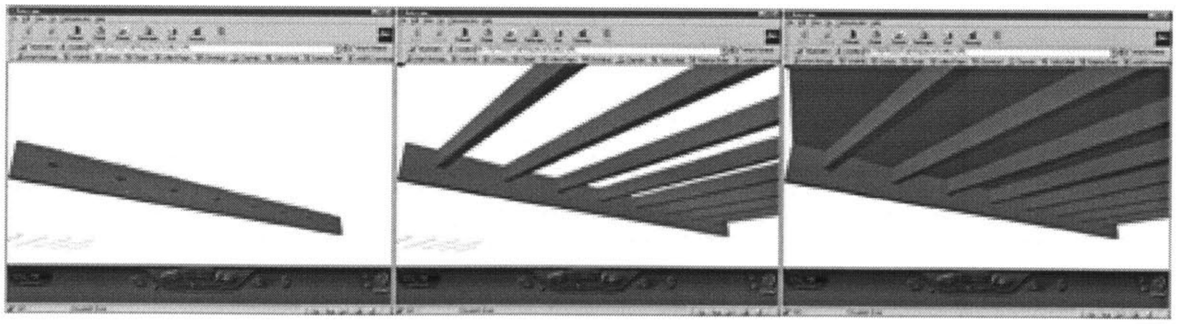


Figure 4—6 Sequence Worlds During Animation

Case Study

As an example of this technology and it's applications I will discuss the work that was done on the High Performance Structures Project within the Master of Engineering Program at the Massachusetts Institute of Technology. The project was to develop conceptual designs for a new Civil and Environmental Engineering Building. The designs began with rough sketches and were taken into a more finalized state where static and dynamic analyses were performed. The architecture of the building was also developed using various visualization techniques. Computer models of various pieces were generated for use in both the visualization models and the analysis models.

To accomplish the visualizations, still renderings were completed, as well as animations of various components. The culmination of the animations was the creation of a building walkthrough featuring the outside and the inside of the design. From the models used to create the renderings, virtual reality models were created to add another level of user interaction. Models were created for the site, including the new building, each of the floor plans and other components. Each model was placed on the project Website to increase their availability to others involved or interested in the project.

In addition to providing links to the virtual worlds, objects within the virtual worlds were specified as hot spots, similar to *TouchSensors,* and linked to other virtual reality files. One example is apparent in the final presentation sample of the virtual reality work. Here, the user begins in the site model, floating high above campus as seen in Figure 4-7. Several views have been provided to allow the user to move around in the world and gain different viewpoints (Figure 4-8) easily without having to move manually. From each of viewpoint, the building is easily visible

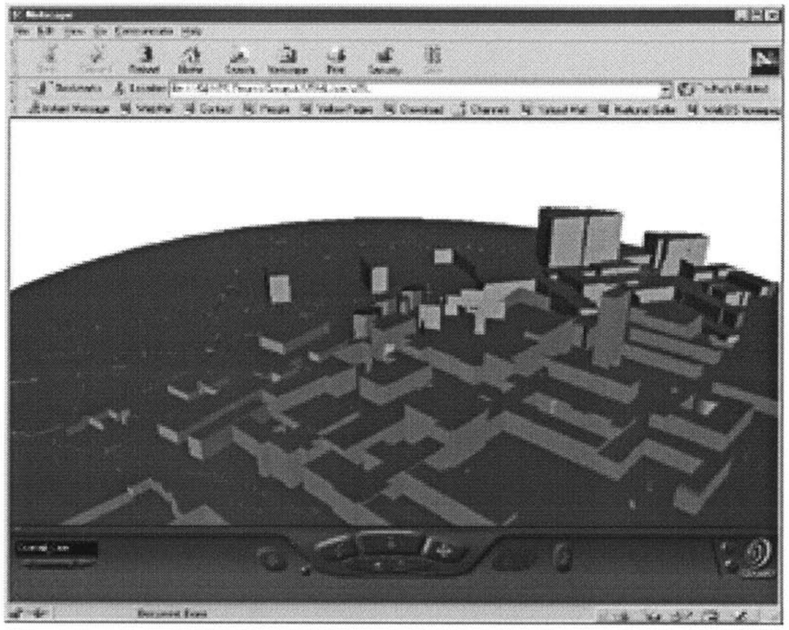


Figure 4—7 Initial View of Site Model

and is actually an active link. When clicked, the user is linked, just as a hyperlink functions in a web page, to another virtual world.

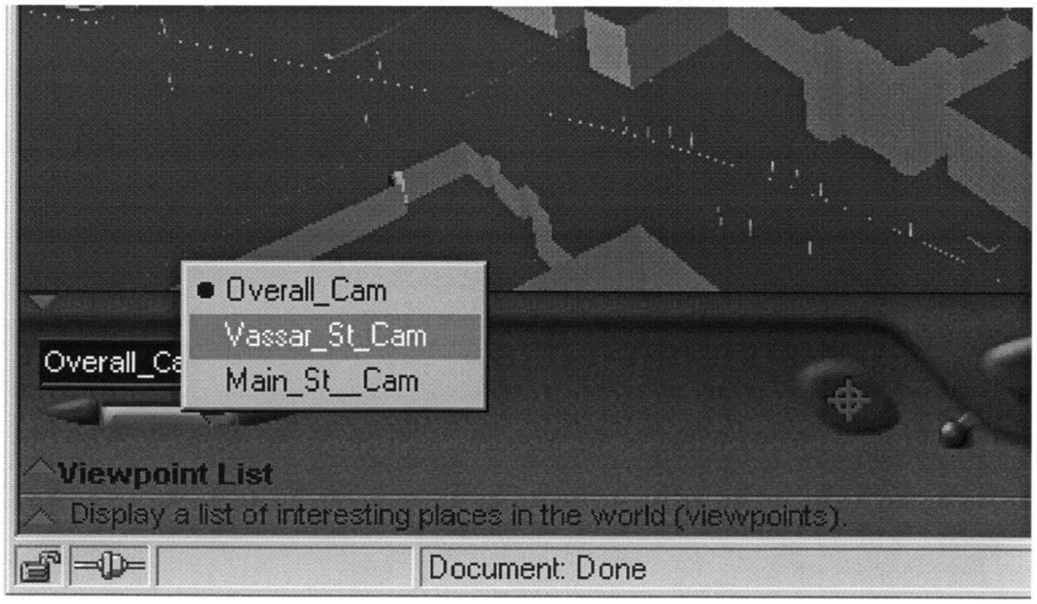


Figure 4—8 Site Model Views

For the purpose of the presentation, the model of the third floor was used as the direct link. For practical uses, the building would be linked to a list of available models, such as floor models. Once in the model of the third floor (Figure 4-9), the user again has several views to choose from, including the typical entrances to the floor: each stairway and the central atrium space elevator. While in this world, the user will notice that there are two active links, one at the location of the central atrium stair and the other at each of the exterior columns. The stair link is quite obvious, and it links to a detailed world of the atrium stair, containing one floor to floor section of the stair that spans between the atrium and the third floor of the building (Figure 4-10). The column link takes the viewer to the world containing a detail cutaway section of the interface that occurs at the column-floor interface (Figure 4-10). This world starts by moving in a circular pattern around the joint to allow the viewer to see it from all angles. The viewer can then move in to particular locations to see the clips that attach the window mullions to the steel framing or any other view they want.

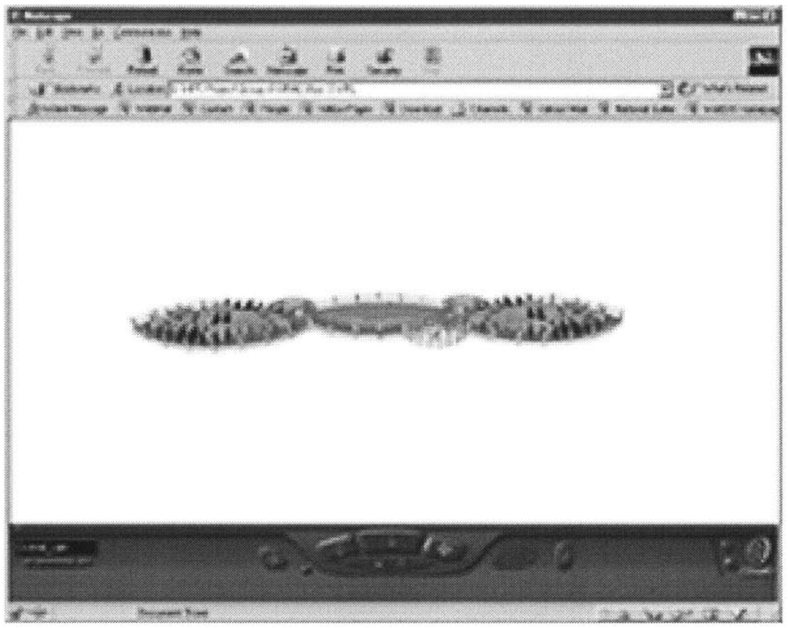
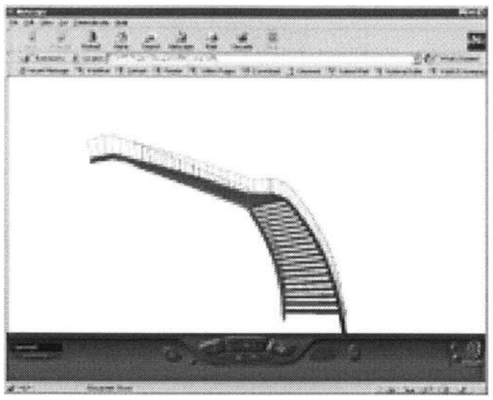


Figure 4—9 3rd Floor Model



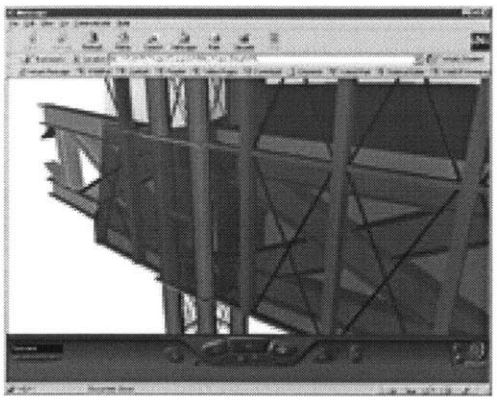


Figure 4—10 Stair and Framing Detail Models

Although this example uses only the built-in components of VRML, it shows its power as a tool to increase information about design issues. By providing these forms of information, the group felt that the viewer had a better sense of the design concept. The format also allows the viewer to see the details of the models used and the lack of detail, in some cases. This can be very useful if a consultant or a client is trying to track the progress of the design. If the model is not updated for quite some time and details are not begin filled in, it is obvious that the design has not been progressing as scheduled. Using the easy to access Web format also provided the greatest portability between systems. Since the VRML technologies used did not take advantage of the new Java specifications, the worlds were easily viewed from any platform using several different VRML browsers.

Chapter 5 - CONCLUSION

Summary

As demonstrated, the VRML can greatly improve existing tools used to convey design ideas and intent. Although this is cutting-edge technology and the format is very unfamiliar to most people involved in the design process, small-scale demonstrations show how beneficial this tool can be. Beginning with examples to aid young design professionals, specific applications were shown that gave the student another tool to visualize what is actually occurring when decisions are made. These demonstration tools can be scaled up to tools more applicable to higher level students, showing more involved concepts, such as building motion.

Often it is difficult to visualize how a building is moving from a three-dimensional stick model in a typical analysis program. This could be greatly augmented using a virtual environment in which the building frame moves along with the envelope giving the student a better idea of how the building is moving. If developed properly, each of these tools can be developed to work within a larger environment of teaching tools, which can be re-used by future teachers and students as components of more advanced virtual worlds.

Although hard to envision, the design applications discussed are entirely possible and have many benefits to offer the profession. However, many designers cannot imagine creating complex virtual worlds for use by the builders and owners of a building. This belief can be attributed to the young age of the American building industry and the speed with which it advances. Europeans, by contrast, are traditionally more technologically advanced builders and are the first to incorporate technology into the design process at any level. As the global market becomes more diverse and more international designers enter the American market, the push to incorporate new technologies will be on the rise.

The need for improved documentation can be justified by the increase in paperwork that is associated with any particular job. Some design consultants hire people to manage the paperwork on a single job. The virtual environment enables this process to become electronic and the tracking and filing of paperwork to become automated, thus creating a smoother process. Additionally, the virtual environment allows the designer to embed information that cannot normally be transmitted via traditional documentation. Since this format is highly Web-based, audio / video recordings can be attached to details, explaining, in the designers words, the intent and how the designer envisioned the building being built. As we move into the information age, the design community will need to redefine it's documentation procedures to allow for the increase in information that is transmitted from the point of design concept to complete design realization.

New Technologi es

Even as we speak, new Internet technologies are being developed. Web-based virtual reality is no exception. As mentioned previously, the VRML specification was released in 1997 and is already being taken over by the new Java 3D API. As I have demonstrated, linking VRML with Java gives the VRML world limitless possibilities. Databases of objects can be embedded into VRML worlds using the Java interface in an empty world.

With the Java 3D API, the power of Java can be taken one step further, taking over the browser's function. This will allow developers to embed a Java applet into an HTML page and allow the user to view the VRML world directly, requiring no plug-in. This will improve the user's experience because only a browser is required. However, this method does require that the developer spend more time on the interface between the user and the world. Java 3D also provides better ways to store VRML objects and to create VRML objects, similar to the creation of objects in object-oriented software design. This will make developing interactive worlds an extension of the typical object-oriented programming paradigm that is prevalent in software design.

One area this package will have problems addressing is the geometrical generation of the virtual worlds. Designers are familiar with high-powered geometrical editors used to generate drawings and three-dimensional models. Using Java3D to create models will be more cumbersome since everything must by input in a text file, and the creation of a geometry editor within Java would be quite an undertaking. For this reason, Java 3D may be limited to using existing geometrical data to create virtual worlds, adding functionality to the components depending on existing properties.

The introduction of Java 3D is a major advancement in the development of tools to aid the virtual-world developer. It is unsure at this time whether Java will continue to be the major programming language for use on the Internet, but if the trend continues the advancement of the Java 3D API will surely take over the VRML specification. The syntax of both languages is similar, as is the process of geometry generation and positioning. Java 3D's major advantage is the underlying programming language that it is based on, to which the VRML cannot compare.

Where do we go from here?

At this time, it is quite obvious that the design profession is behind the current technology available. In the coming years, the profession as a whole will need to incorporate more information technologies into their daily routines. As the Internet has developed, business has been changed, allowing companies to conduct business without a word between them. The business of design is much more people driven, as the decisions made affect the appearance of a physical object in our world. However, the daily practice of information sharing can be greatly augmented by utilizing emerging computer technologies.

Virtual reality is just a step away from where the profession is currently. Architecture offices have begun to utilize the power of three-dimensional computer modeling to augment their design tools. Engineers use sophisticated design programs to analyze structures in minutes, rather than days. The next step in static three-dimensional worlds is to add interaction, providing real-time response to user action. If a couple of nicely rendered still images of a building help visualize a building design, imagine what an interactive model could achieve where the user is in control of the rendered view.

In conclusion, with its portable format and the ability to embed a highly function programming language, the Virtual Reality Modeling Language specification is a tool to improve design communication. Although the VRML format is generalized and not specific to the building industry, the profession can see the potential of use of such systems to the design environment. Ultimately, the profession will need to adopt their own standards to present and design buildings with some, if not all, the features incorporated into existing web-based virtual reality formats.

REFERENCES

Ames, Nadeau and Moreland. VRML 2.0 Sourcebook. Wiley and Sons Inc., New York, 1997.

James Gere and Stephen Timoshenko. Mechanics of Materials, Third Edition. PWD-Kent Publishing Company, Boston, 1990.

Lea, Rodger. *Java and VRML 2.0 Part 1: Basic Theory*. Available: http://www.vrmlsite.com/feb97/a.cgi/spot2.html. 12 April 1999.

Lea, Rodger. *Java and VRML 2.0 Part 2: Putting Theory into Practice*. Available: http://www.vrmlsite.com/mar97/a.cgi/spot3.html. 12 April 1999.

Rikk Carey, Gavin Bell, Chris Marrin. *ISO/IEC 14772-1:1997, Virtual Reality Modeling Language, (VRML97)*. Available: http://www.vrml.org/Specification/VRML97. 12 April 1999.

Scott, Adrian. *The Marriage of Java and VRML*. Available: http://www.vrmlsite.com/sep96/spotlight/javavrml/javavrml.html. 12 April 1999.

Appendix A – SOURCE FILES

General

The content of this paper is based around the four examples discussed and the one case study. The Java source code has been reproduced here for greater detail and complete syntax on how to link the two technologies together. The Virtual Reality files are quite large, as the method for describing even simple geometry can be quite complex. I will include portions of the files that are pertinent to the linking of Java and VRML. Much of the geometry has been edited out to keep the information to a manageable amount.

Space Relationship Example

VRML – “twoRooms.wrl”

```
#VRML V2.0 utf8

WorldInfo {…}
NavigationInfo {…}
DEF Plan_View Viewpoint {…}
DEF Room1 Viewpoint {…}
DEF Room2 Viewpoint {…}
DEF wall_app Appearance {…}
DEF int_app Appearance {…}
DEF Floor Transform {…}
DEF LeftWall Transform {…}
DEF RightWall Transform {…}
DEF RearWall Transform {…}
DEF FrontWall Transform {…}

DEF InteriorWall1 Transform {
  …
    DEF Wall1Sensor TouchSensor {}
}

DEF InteriorWall2 Transform {
  …
    DEF Wall2Sensor TouchSensor {}
}

DEF OneRoomControl Script {
        directOutput TRUE

        field SFNode w1 USE InteriorWall1
        field SFNode w2 USE InteriorWall2

        eventIn SFBool clicked1
        eventIn SFBool clicked2

        url "moveWall.class"
}

ROUTE Wall1Sensor.isActive          TO OneRoomControl.clicked1
ROUTE Wall2Sensor.isActive          TO OneRoomControl.clicked2
```

Java – “moveWall.java”

```
import vrml.*;
import vrml.field.*;
import vrml.node.*;

public class moveWall extends Script{
        private SFNode w1, w2;

        public void initialize() {
                        //Connect java nodes with vrml nodes
                        w1 = (SFNode) getField("w1");
                        w2 = (SFNode) getField("w2");
        }

        public void processEvent(Event e) {
                ConstSFBool v = (ConstSFBool)e.getValue();
                //Only perform the movement on mouse click, not release
                if (v.getValue()) {
                        SFVec3f t1, t2;
                        t1 =
(SFVec3f)((Node)w1.getValue()).getExposedField("translation");
                        t2 =
(SFVec3f)((Node)w2.getValue()).getExposedField("translation");
                        if (e.getName().equals("clicked1")==true) {
                                t1.setValue(t1.getX()+10.0f, t1.getY(),
t1.getZ());
                                t2.setValue(t2.getX()+10.0f, t2.getY(),
t2.getZ());
                        }else if (e.getName().equals("clicked2")==true) {
                                t1.setValue(t1.getX()-10.0f, t1.getY(),
t1.getZ());
                                t2.setValue(t2.getX()-10.0f, t2.getY(),
t2.getZ());
                        }
                }
        }
}
```

Simple Beam Column Example

VRML – "Span.wrl"

```
#VRML V2.0 utf8

WorldInfo {…}
NavigationInfo {…}
DEF View1 Viewpoint {…}
DEF Column1 Transform {…}
DEF Column2 Transform {…}

DEF Beam1 Transform {
  …
    DEF sizer TouchSensor {}
}

DEF SizeSpan Script {
        field SFNode c1 USE Column1
        field SFNode c2 USE Column2
        field SFNode b1 USE Beam1

        eventIn SFBool size

        directOutput TRUE

        url "sizeBeam.class"
}

ROUTE sizer.isActive          TO SizeSpan.size
```

Java – "sizeBeam.java"

```
import vrml.*;
import vrml.field.*;
import vrml.node.*;
import java.awt.event.*;

public class sizeBeam extends Script implements ActionListener,
AdjustmentListener {
        private trNode col1, col2, bm1; //Nodes to get data from VRML file
        private Binterface bint;            //Interface to provide input

        public void initialize() {
                col1 = new trNode((SFNode) getField("c1"));
                col2 = new trNode((SFNode) getField("c2"));
                bm1 = new trNode((SFNode) getField("b1"));

                bint = new Binterface(this);

                this.resize();
        }
        public void shutdown() {
                bint.dispose();
        }

        //Scrollbar listeners to catch when scrollbar events...
        public void adjustmentValueChanged(AdjustmentEvent ae) {
                this.resize();
        }
        //Action Listeners to catch button events...
        public void actionPerformed(ActionEvent ae) {
                if (ae.getSource().equals(bint.getIncrease())) {
                        col1.moveZ(-5f);
                        col2.moveZ(5f);
                        this.resize();
                }else if (ae.getSource().equals(bint.getDecrease())) {
                        col1.moveZ(5f);
                        col2.moveZ(-5f);
                        this.resize();
                }
        }
        public void resize() {
                float w, l, E, h2b;
                float tau, sigma, delta;
                float b, b1, b2, b3;
                w = bint.getLoad().getValue();
                l = (col1.getdist(col2)).getZ();
                E = bint.getModulus().getValue();
                h2b = bint.getRatio().getValue();

                tau = bint.getShearStress().getValue();
                sigma = bint.getBendingStress().getValue();
                delta = bint.getDeflection().getValue();
                //check shear
                b1 = (float)Math.pow((3 * w * l) / (h2b * 4 * tau), (1.0 /
2.0));
                //check bending
                b2 = (float)Math.pow(((3 * w * Math.pow(l, 2)) / (4 *
Math.pow(h2b, 2) * sigma)), (1.0 / 3.0));
                //check deflection
                b3 = (float)Math.pow(((60 * w * Math.pow(l, 3) * delta) /
(384 * E * Math.pow(h2b, 3))), (1.0 / 4.0));
                if (b1 > Math.max(b2, b3)) {
                        bint.writeCase("Shear");
                }else if (b2 > Math.max(b1, b3)) {
                        bint.writeCase("Bending");
                }else if (b3 > Math.max(b1, b2)) {
                        bint.writeCase("Deflection");
                }
                b = Math.max(b1, Math.max(b2, b3));

                System.out.println("Sizing Data");
                System.out.println("Load = " + w + "  Length = " + l);
                System.out.println("E = " + E + "  H/B Ratio = " + h2b);
                System.out.println("Max Shear Stress = " + tau + "  and b =
" + b1);
                System.out.println("Max Bending Stress = " + sigma + "  and
b = " + b2);
```

```
                System.out.println("Max Deflection = L/" + delta + "  and b
= " + b3);

                bm1.setlocation(0, 100 + b * h2b / 2, 0);
                bm1.scaleX(b/2);                        //Width of beam
                bm1.scaleY(b * h2b / 2);        //Height of beam
                bm1.scaleZ(1/2);                        //Respan beam

        }
        //Catch the vrml events...
        public void processEvent(Event e) {
                ConstSFBool v = (ConstSFBool)e.getValue();
                //Only perform the movement on mouse click, not release
                if (v.getValue()) {
                        bint.show();
                }
        }
}
```

Java – “Binterface.java”

```
import java.awt.*;
import java.awt.event.*;

public class Binterface extends Frame implements WindowListener
{
        private sizeBeam sB;  //Reference to main class

        private Label lLoad, lRatio, lModulus, lShearStress,
lBendingStress, lDeflection, lCase, lCase2;
        private String sLoad, sRatio, sModulus, sShearStress,
sBendingStress, sDeflection, sCase;
        private Scrollbar scbLoad, scbRatio, scbModulus, scbShearStress,
scbBendingStress, scbDeflection;
        private Button btnIncrease, btnDecrease;

        public Binterface(sizeBeam s)
        {
                sB = new sizeBeam();
                sB = s;

                //Initialize gui objects
                sLoad = new String("Load: ");
                sRatio = new String("Height/Width Ratio: ");
                sModulus = new String("Modulus of Elasticity: ");
                sShearStress = new String("Allowable Shear Stress: ");
                sBendingStress = new String("Allowable Bending Stress: ");
                sDeflection = new String("Allowable Deflection: ");
                sCase = new String("Controlling Case: ");

                lLoad = new Label(sLoad);
                lRatio = new Label(sRatio);
                lModulus = new Label(sModulus);
                lShearStress = new Label(sShearStress);
                lBendingStress = new Label(sBendingStress);
                lDeflection = new Label(sDeflection);
                lCase = new Label(sCase);
                lCase2 = new Label("");

                scbLoad = new Scrollbar(Scrollbar.HORIZONTAL, 5000, 500,
1000, 10000);
                        scbLoad.setUnitIncrement(500);
                scbRatio = new Scrollbar(Scrollbar.HORIZONTAL, 1, 1, 1, 5);
                        scbRatio.setUnitIncrement(1);
                scbModulus = new Scrollbar(Scrollbar.HORIZONTAL, 30000000,
5000000, 10000000, 40000000);
                        scbModulus.setUnitIncrement(5000000);
```

```
		scbShearStress = new Scrollbar(Scrollbar.HORIZONTAL, 15000,
500, 10000, 20000);
			scbShearStress.setUnitIncrement(500);
		scbBendingStress = new Scrollbar(Scrollbar.HORIZONTAL,
20000, 1000, 10000, 30000);
			scbBendingStress.setUnitIncrement(1000);
		scbDeflection = new Scrollbar(Scrollbar.HORIZONTAL, 350,
50, 100, 1000);
			scbDeflection.setUnitIncrement(50);

		btnDecrease = new Button("Decrease Span");
		btnIncrease = new Button("Increase Span");

		//Set up gui
		this.setSize(300, 350);
		this.setTitle("Beam Sizing Example");
		this.setLayout(new GridLayout(11, 2));

		this.add(new Label("Beam Inputs"));
		this.add(new Label());
		this.add(lLoad);
		this.add(scbLoad);
		this.add(lRatio);
		this.add(scbRatio);
		this.add(lModulus);
		this.add(scbModulus);

		this.add(new Label("Design Values"));
		this.add(new Label());
		this.add(lShearStress);
		this.add(scbShearStress);
		this.add(lBendingStress);
		this.add(scbBendingStress);
		this.add(lDeflection);
		this.add(scbDeflection);

		this.add(new Label("Change column positions"));
		this.add(new Label());
		this.add(btnDecrease);
		this.add(btnIncrease);
		this.add(lCase);
		this.add(lCase2);

		//Add listeners...
		this.addWindowListener(this);
		scbLoad.addAdjustmentListener(sB);
		scbRatio.addAdjustmentListener(sB);
		scbModulus.addAdjustmentListener(sB);
		scbShearStress.addAdjustmentListener(sB);
		scbBendingStress.addAdjustmentListener(sB);
		scbDeflection.addAdjustmentListener(sB);
		btnDecrease.addActionListener(sB);
		btnIncrease.addActionListener(sB);

		//show the form
		this.show();
	}
	public void shutdown() {
		this.setVisible(false);
	}
	public Scrollbar getLoad() {
		return scbLoad;
	}
	public Scrollbar getRatio() {
		return scbRatio;
	}
	public Scrollbar getModulus() {
		return scbModulus;
	}
	public Scrollbar getShearStress() {
```

```
                return scbShearStress;
        }
        public Scrollbar getBendingStress() {
                return scbBendingStress;
        }
        public Scrollbar getDeflection() {
                return scbDeflection;
        }
        public Button getDecrease() {
                return btnDecrease;
        }
        public Button getIncrease() {
                return btnIncrease;
        }
        public void writeCase(String s) {
                lCase2.setText(s);
        }
        //Listener Methods...
        public void windowClosing(WindowEvent e) {
                this.setVisible(false);
        }
        public void windowClosed(WindowEvent e) {}
        public void windowOpened(WindowEvent e) {}
        public void windowIconified(WindowEvent e) {}
        public void windowDeiconified(WindowEvent e) {}
        public void windowActivated(WindowEvent e) {}
        public void windowDeactivated(WindowEvent e) {}
}
```

Java – "trNode.java"

```
import vrml.node.*;
import vrml.field.*;

public class trNode
{
        private SFNode n;
        private SFVec3f t, s;

        public trNode(SFNode nn) {
                n = nn;
                t = (SFVec3f)
((Node)n.getValue()).getExposedField("translation");
                s = (SFVec3f)
((Node)n.getValue()).getExposedField("scale");
        }
        public SFVec3f getTranslation(){
                return this.t;
        }
        public SFVec3f getScale(){
                return this.s;
        }
        public void setlocation(float x, float y, float z) {
                t.setValue(x, y, z);
        }
        public void setscale(float xs, float ys, float zs) {
                s.setValue(xs, ys, zs);
        }
        public SFVec3f getdist(trNode n2) {
                SFVec3f d;
                d = new SFVec3f(n2.getTranslation().getX()-t.getX(),
n2.getTranslation().getY()-t.getY(), n2.getTranslation().getZ()-t.getZ());
                return d;
        }
        public void moveX(float f) {
                t.setValue(t.getX()+f, t.getY(), t.getZ());
        }
        public void moveY(float f) {
                t.setValue(t.getX(), t.getY()+f, t.getZ());
        }
        public void moveZ(float f) {
                t.setValue(t.getX(), t.getY(), t.getZ()+f);
        }
        public void scaleX(float f) {
                s.setValue(f, s.getY(), s.getZ());
        }
        public void scaleY(float f) {
                s.setValue(s.getX(), f, s.getZ());
        }
        public void scaleZ(float f) {
                s.setValue(s.getX(), s.getY(), f);
        }
}
```

Design Details Example

VRML – "DesignDetails.wrl"

```
#VRML V2.0 utf8

WorldInfo {…}
NavigationInfo {…}
DEF Default_View Viewpoint {…}

DEF BEAM Transform {
  …
    DEF beamSNS TouchSensor {}
}
DEF RCOLUMN Transform {
  …
    DEF rcolumnSNS TouchSensor {}
}
DEF LCOLUMN Transform {
  …
    DEF lcolumnSNS TouchSensor {}
}
DEF RPLATE Transform {
  …
    DEF rplateSNS TouchSensor {}
}
DEF RBOLT1 Transform {
  …
    DEF rbolt1SNS TouchSensor {}
}
DEF RBOLT2 Transform {
  …
    DEF rbolt2SNS TouchSensor {}
}
DEF RBOLT3 Transform {
  …
    DEF rbolt3SNS TouchSensor {}
}
DEF LPLATE Transform {
  …
    DEF lplateSNS TouchSensor {}
}
DEF LBOLT1 Transform {
  …
    DEF lbolt1SNS TouchSensor {}
}
```

```
DEF LBOLT2 Transform {
  ...
    DEF lbolt2SNS TouchSensor {}
}
DEF LBOLT3 Transform {
  ...
    DEF lbolt3SNS TouchSensor {}
}

DEF ddControl Script {

        eventIn SFBool Lcol
        eventIn SFBool Rcol
        eventIn SFBool bm
        eventIn SFBool Lpl
        eventIn SFBool Rpl
        eventIn SFBool Lb1
        eventIn SFBool Lb2
        eventIn SFBool Lb3
        eventIn SFBool Rb1
        eventIn SFBool Rb2
        eventIn SFBool Rb3

        url "DDinfo.class"
}

ROUTE lcolumnSNS.isOver               TO ddControl.Lcol
ROUTE rcolumnSNS.isOver               TO ddControl.Rcol
ROUTE beamSNS.isOver          TO ddControl.bm
ROUTE lplateSNS.isOver        TO ddControl.Lpl
ROUTE rplateSNS.isOver        TO ddControl.Rpl
ROUTE lbolt1SNS.isOver        TO ddControl.Lb1
ROUTE lbolt2SNS.isOver        TO ddControl.Lb2
ROUTE lbolt3SNS.isOver        TO ddControl.Lb3
ROUTE rbolt1SNS.isOver        TO ddControl.Rb1
ROUTE rbolt2SNS.isOver        TO ddControl.Rb2
ROUTE rbolt3SNS.isOver        TO ddControl.Rb3
```

Java – "DDinfo.java"

```
import vrml.*;
import vrml.field.*;
import vrml.node.*;

public class DDinfo extends Script{
        private DDinterface info;

        public void initialize() {
                info = new DDinterface();
        }
        public void processEvent(Event e) {
                ConstSFBool v = (ConstSFBool)e.getValue();
                //Only perform the movement on mouse click, not release
                if (v.getValue()) {
                        info.show();
                        String ename;
                        ename = e.getName();
                        if (ename.equals("Lpl")==true) {
                                info.write("Left Plate", "Steel",
"PL11x10x0.5", "Typical shear Plate");
                        }else if (ename.equals("Rpl")==true) {
                                info.write("Right Plate", "Steel",
"PL11x10x0.5", "Typical shear Plate");
                        }else if (ename.equals("Lcol")==true) {
                                info.write("Left Column", "Steel", "W10x49",
"120 inches long");
                        }else if (ename.equals("Rcol")==true) {
                              info.write("Rigth Column", "Steel",
"W10x49", "120 inches long");
                        }else if (ename.equals("bm")==true) {
                              info.write("Only Beam", "Steel", "W16x31",
"108 inches long");
                        }else if (ename.equals("Lb1")==true) {
                              info.write("Bolt", "A325 Steel", "3/4in.
Diameter", "Typical thru bolt");
                        }else if (ename.equals("Lb2")==true) {
                              info.write("Bolt", "A325 Steel", "3/4in.
Diameter", "Typical thru bolt");
                        }else if (ename.equals("Lb3")==true) {
                              info.write("Bolt", "A325 Steel", "3/4in.
Diameter", "Typical thru bolt");
                        }else if (ename.equals("Rb1")==true) {
                              info.write("Bolt", "A325 Steel", "3/4in.
Diameter", "Typical thru bolt");
                        }else if (ename.equals("Rb2")==true) {
                              info.write("Bolt", "A325 Steel", "3/4in.
Diameter", "Typical thru bolt");
                        }else if (ename.equals("Rb3")==true) {
                              info.write("Bolt", "A325 Steel", "3/4in.
Diameter", "Typical thru bolt");
                        }
                  }else {
                        info.write("", "", "", "");
                  }
            }
      }
```

Java – "DDinterface.java"

```
import java.awt.*;
import java.awt.event.*;

public class DDinterface extends Frame implements WindowListener
{
        private Label lname, lmaterial, lsize, linfo;
        private String txtName, txtMaterial, txtSize, txtInfo;

        public DDinterface()
        {
                txtName = new String("Name: ");
                txtMaterial = new String("Material: ");
                txtSize = new String("Size: ");
                txtInfo = new String("Other Info: ");
                this.addWindowListener(this);
                this.setSize(200, 150);
                this.setTitle("Design Details Example");
                this.setLayout(new GridLayout(4,1));
                lname = new Label(txtName);
                lmaterial = new Label(txtMaterial);
                lsize = new Label(txtSize);
                linfo = new Label(txtInfo);
                this.add(lname);
                this.add(lmaterial);
                this.add(lsize);
                this.add(linfo);
                this.show();
        }
        public void write(String nm, String ma, String sz, String in) {
                lname.setText(txtName + nm);
                lmaterial.setText(txtMaterial + ma);
                lsize.setText(txtSize + sz);
                linfo.setText(txtInfo + in);
        }
        public void windowClosing(WindowEvent e) {
               this.setVisible(false);
        }
        public void windowClosed(WindowEvent e) {}
        public void windowOpened(WindowEvent e) {}
        public void windowIconified(WindowEvent e) {}
        public void windowDeiconified(WindowEvent e) {}
        public void windowActivated(WindowEvent e) {}
        public void windowDeactivated(WindowEvent e) {}
}
```

Process Animati on Example

VRML – "studfloor.wrl"

```
#VRML V2.0 utf8

WorldInfo {…}
NavigationInfo {…}
DEF Overall_View Viewpoint {…}

DEF BEAM1 Transform {
  …
}
ROUTE BEAM1-TIMER.fraction_changed TO BEAM1-POS-INTERP.set_fraction
ROUTE BEAM1-POS-INTERP.value_changed TO BEAM1.set_translation

DEF HANGER Transform {
  …
}
ROUTE HANGER-TIMER.fraction_changed TO HANGER-POS-INTERP.set_fraction
ROUTE HANGER-POS-INTERP.value_changed TO HANGER.set_translation

DEF JOIST1 Transform {
  …
}
ROUTE JOIST1-TIMER.fraction_changed TO JOIST1-POS-INTERP.set_fraction
ROUTE JOIST1-POS-INTERP.value_changed TO JOIST1.set_translation

DEF JOIST2 Transform {
  …
}
ROUTE JOIST2-TIMER.fraction_changed TO JOIST2-POS-INTERP.set_fraction
ROUTE JOIST2-POS-INTERP.value_changed TO JOIST2.set_translation

DEF JOIST3 Transform {
  …
}
ROUTE JOIST3-TIMER.fraction_changed TO JOIST3-POS-INTERP.set_fraction
ROUTE JOIST3-POS-INTERP.value_changed TO JOIST3.set_translation

DEF JOIST4 Transform {
  …
}
ROUTE JOIST4-TIMER.fraction_changed TO JOIST4-POS-INTERP.set_fraction
ROUTE JOIST4-POS-INTERP.value_changed TO JOIST4.set_translation

DEF JOIST5 Transform {
  …
}
ROUTE JOIST5-TIMER.fraction_changed TO JOIST5-POS-INTERP.set_fraction
ROUTE JOIST5-POS-INTERP.value_changed TO JOIST5.set_translation
```

```
DEF JOIST6 Transform {
  ...
}

ROUTE JOIST6-TIMER.fraction_changed TO JOIST6-POS-INTERP.set_fraction
ROUTE JOIST6-POS-INTERP.value_changed TO JOIST6.set_translation

DEF JOIST7 Transform {
  ...
}
ROUTE JOIST7-TIMER.fraction_changed TO JOIST7-POS-INTERP.set_fraction
ROUTE JOIST7-POS-INTERP.value_changed TO JOIST7.set_translation

DEF JOIST8 Transform {
  ...
}
ROUTE JOIST8-TIMER.fraction_changed TO JOIST8-POS-INTERP.set_fraction
ROUTE JOIST8-POS-INTERP.value_changed TO JOIST8.set_translation

DEF JOIST9 Transform {
  ...
}
ROUTE JOIST9-TIMER.fraction_changed TO JOIST9-POS-INTERP.set_fraction
ROUTE JOIST9-POS-INTERP.value_changed TO JOIST9.set_translation

DEF SEALER Transform {
  ...
}
ROUTE SEALER-TIMER.fraction_changed TO SEALER-POS-INTERP.set_fraction
ROUTE SEALER-POS-INTERP.value_changed TO SEALER.set_translation

DEF PLYWOOD1 Transform {
  ...
}
ROUTE PLYWOOD1-TIMER.fraction_changed TO PLYWOOD1-POS-INTERP.set_fraction
ROUTE PLYWOOD1-POS-INTERP.value_changed TO PLYWOOD1.set_translation

DEF PLYWOOD2 Transform {
  ...
}
ROUTE PLYWOOD2-TIMER.fraction_changed TO PLYWOOD2-POS-INTERP.set_fraction
ROUTE PLYWOOD2-POS-INTERP.value_changed TO PLYWOOD2.set_translation

DEF PLYWOOD3 Transform {
  ...
}
ROUTE PLYWOOD3-TIMER.fraction_changed TO PLYWOOD3-POS-INTERP.set_fraction
ROUTE PLYWOOD3-POS-INTERP.value_changed TO PLYWOOD3.set_translation

DEF PLYWOOD4 Transform {
  ...
}
ROUTE PLYWOOD4-TIMER.fraction_changed TO PLYWOOD4-POS-INTERP.set_fraction
ROUTE PLYWOOD4-POS-INTERP.value_changed TO PLYWOOD4.set_translation

DEF NAILS Transform {
  ...
}
ROUTE NAILS-TIMER.fraction_changed TO NAILS-POS-INTERP.set_fraction
ROUTE NAILS-POS-INTERP.value_changed TO NAILS.set_translation

DEF Sequence Script {
        field SFNode b1t USE BEAM1-TIMER
        field SFNode h1t USE HANGER-TIMER
        field SFNode j1t USE JOIST1-TIMER
        field SFNode j2t USE JOIST2-TIMER
        field SFNode j3t USE JOIST3-TIMER
        field SFNode j4t USE JOIST4-TIMER
        field SFNode j5t USE JOIST5-TIMER
        field SFNode j6t USE JOIST6-TIMER
        field SFNode j7t USE JOIST7-TIMER
        field SFNode j8t USE JOIST8-TIMER
        field SFNode j9t USE JOIST9-TIMER
        field SFNode s1t USE SEALER-TIMER
        field SFNode p1t USE PLYWOOD1-TIMER
        field SFNode p2t USE PLYWOOD2-TIMER
        field SFNode p3t USE PLYWOOD3-TIMER
        field SFNode p4t USE PLYWOOD4-TIMER
        field SFNode n1t USE NAILS-TIMER

        eventIn SFFloat b
        eventIn SFFloat h
        eventIn SFFloat j1
        eventIn SFFloat j2
        eventIn SFFloat j3
        eventIn SFFloat j4
        eventIn SFFloat j5
        eventIn SFFloat j6
        eventIn SFFloat j7
        eventIn SFFloat j8
        eventIn SFFloat j9
        eventIn SFFloat s
        eventIn SFFloat p1
        eventIn SFFloat p2
        eventIn SFFloat p3
        eventIn SFFloat p4
        eventIn SFFloat n

        directOutput TRUE

        url "sequence.class"
}
ROUTE BEAM1-TIMER.fraction_changed              TO Sequence.b
ROUTE HANGER-TIMER.fraction_changed             TO Sequence.h
ROUTE JOIST1-TIMER.fraction_changed             TO Sequence.j1
ROUTE JOIST2-TIMER.fraction_changed             TO Sequence.j2
ROUTE JOIST3-TIMER.fraction_changed             TO Sequence.j3
ROUTE JOIST4-TIMER.fraction_changed             TO Sequence.j4
ROUTE JOIST5-TIMER.fraction_changed             TO Sequence.j5
ROUTE JOIST6-TIMER.fraction_changed             TO Sequence.j6
ROUTE JOIST7-TIMER.fraction_changed             TO Sequence.j7
ROUTE JOIST8-TIMER.fraction_changed             TO Sequence.j8
ROUTE JOIST9-TIMER.fraction_changed             TO Sequence.j9
ROUTE SEALER-TIMER.fraction_changed             TO Sequence.s
ROUTE PLYWOOD1-TIMER.fraction_changed       TO Sequence.p1
ROUTE PLYWOOD2-TIMER.fraction_changed       TO Sequence.p2
ROUTE PLYWOOD3-TIMER.fraction_changed       TO Sequence.p3
ROUTE PLYWOOD4-TIMER.fraction_changed       TO Sequence.p4
ROUTE NAILS-TIMER.fraction_changed              TO Sequence.n
```

Java – "sequence.java"

```
import vrml.*;
import vrml.field.*;
import vrml.node.*;
import java.awt.*;
import java.awt.event.*;

public class sequence extends Script implements ActionListener {
        private Frame f;
        private Button bt;
        private boolean pause;
        private SFNode[] timers;

        public void initialize() {
                pause = false;
                timers = new SFNode[17];

                timers[0] = (SFNode)getField("b1t");
                timers[1] = (SFNode)getField("h1t");
                timers[2] = (SFNode)getField("j1t");
                timers[3] = (SFNode)getField("j2t");
                timers[4] = (SFNode)getField("j3t");
                timers[5] = (SFNode)getField("j4t");
                timers[6] = (SFNode)getField("j5t");
                timers[7] = (SFNode)getField("j6t");
                timers[8] = (SFNode)getField("j7t");
                timers[9] = (SFNode)getField("j8t");
                timers[10] = (SFNode)getField("j9t");
                timers[11] = (SFNode)getField("s1t");
                timers[12] = (SFNode)getField("p1t");
                timers[13] = (SFNode)getField("p2t");
                timers[14] = (SFNode)getField("p3t");
                timers[15] = (SFNode)getField("p4t");
                timers[16] = (SFNode)getField("n1t");

                f = new Frame();
                f.setSize(100, 100);
                f.setTitle("Stop/Start Animation");
                f.setLayout(new FlowLayout());
                bt = new Button();
                bt.setLabel("Pause");
                bt.addActionListener(this);
                f.add(bt);
                f.show();
        }
        public void shutdown() {
                f.dispose();
        }

        public void actionPerformed(ActionEvent e) {
                SFBool sfb;
                if (pause==true) {     //Paused, now restart
                        pause = false;
                        for (int i = 0; i<17; i++) {
                                sfb = (SFBool)
((Node)timers[i].getValue()).getExposedField("enabled");
                                sfb.setValue(true);
                        }
                        bt.setLabel("Pause");
                }else {                        //Running, now stop
                        pause = true;
                        for (int i = 0; i<17; i++) {
                                sfb = (SFBool)
((Node)timers[i].getValue()).getExposedField("enabled");
                                sfb.setValue(false);
                        }
                        bt.setLabel("Restart");
                }
        }
        /*
        public void processEvent(vrml.Event e) {
                //if ((ConstSFFloat)e.getValue()==new ConstSFFloat(1.0f)) {
                if (e.getName().equals("b")==true) {
                        System.out.println("Placing Beam");
                }else if (e.getName().equals("h")==true) {
                        System.out.println("Attaching joist hangers to
beam");
                }else if (e.getName().equals("j1")==true) {
                        System.out.println("Placing Joist #1");
                }else if (e.getName().equals("j2")==true) {
                        System.out.println("Placing Joist #2");
                }else if (e.getName().equals("j3")==true) {
                        System.out.println("Placing Joist #3");
                }else if (e.getName().equals("j4")==true) {
                        System.out.println("Placing Joist #4");
                }else if (e.getName().equals("j5")==true) {
                        System.out.println("Placing Joist #5");
                }else if (e.getName().equals("j6")==true) {
                        System.out.println("Placing Joist #6");
                }else if (e.getName().equals("j7")==true) {
                        System.out.println("Placing Joist #7");
                }else if (e.getName().equals("j8")==true) {
                        System.out.println("Placing Joist #8");
                }else if (e.getName().equals("j9")==true) {
                        System.out.println("Placing Joist #9");
                }else if (e.getName().equals("s")==true) {
                        System.out.println("Laying Caulking bead");
                }else if (e.getName().equals("p1")==true) {
                        System.out.println("Placing plywood");
                }else if (e.getName().equals("p2")==true) {
                        System.out.println("Placing plywood");
                }else if (e.getName().equals("p3")==true) {
                        System.out.println("Placing plywood");
                }else if (e.getName().equals("p4")==true) {
                        System.out.println("Placing plywood");
                }else if (e.getName().equals("n")==true) {
                        System.out.println("Attaching plywood with nails");
                }
//              }
        }
        */
}
```

Appendix B – SOFTWARE LIST

Software List

The following software was used to complete the design and viewing of the interactive virtual environments in this paper.

VRML World Creation

- AutoCAD V14.0
- 3D Studio MAX v2.5 and 3D Studio VIZ 2.0

Java and VRML source code editing and compilation

- Microsoft Visual J++ 6.0
- Sun Java Development Kit 1.2

Internet Browsers and Viewers

Microsoft Internet Explorer 4.0

Netscape Navigator 4.05

Sony community Place 2.0

CosmoSoftware Cosmo Player 2.1